Integration of and Access to Distributed Data and Tools in Genomics

Hajar Sadrarhami

A thesis

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements

For the Degree of Master of Computer Science

Concordia University

Montréal, Québec, Canada

April 2011

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: Hajar Sadrarhami

Entitled: Integration of and Access to Distributed Data and Tools in Genomics

and submitted in partial fulfillment of the requirements for the degree of

Master of Computer Science

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining commitee:

______________________ Chair
Dr. Nematollaah Shiri

______________________ Examiner
Dr. Volker Haarslev

______________________ Examiner
Dr. Clement Lam

______________________ Supervisor
Dr. Gregory Butler

Approved by ______________________
Chair of Department or Graduate Program Director

__________ 20 ______ ______________________

Dr. Robin Drew, Dean

Faculty of Engineering and Computer Science

Abstract

Integration of and Access to Distributed Data and Tools in Genomics

Hajar Sadrarhami

One of the important data sources in bioinformatics is protein or nucleotide sequences that are used as input to many programs to collectively or individually analyze them. There exists an ample amount of protein sequences scattered over many different databases. This division complicates the process of feeding them into existing programs to be further analyzed. Moreover, there exists a program integration portal, namely Mobyle that makes the common programs available with unified interface to the users; in addition, it provides the functionality of chaining the results from one program to another. The two existing programs in Mobyle fetch sequences to feed the other programs, however, they fetch sequences from limited number of databases that are statically defined by the Mobyle administrator. In addition, neither of these tools have access to the DAS servers, resulting in the loss of a major data source. In this work, a program was developed and integrated, namely DasSeqFetcher, for use in Mobyle to dynamically fetch sequences from all available sequence databases providing a DAS reference server. Also, both DAS reference and annotation servers were developed for a database made by our research group which holds experimentally characterized lignocellulose-active proteins. The reference servers can then be added to DAS registry to be used by DAS client tools, e.g. DasSeqFetcher.

Acknowledgments

I would like to express my heartfelt appreciation to my supervisor, Professor. Gregory Butler for his supports, insights, reviews and professional advice. Without his help this work would not be possible.

I would like to express my gratitude for all my colleagues in the Cellulosic Biofuel Network Project for their great contribution.

Special thanks go out to my friends Christine Kehyayan, Stephen Barrett, Jianlong Qi and Aseel Hmood for their valuable comments, discussions and support. Also, I appreciate Ms. Shira Katz for her editorial assistance.

Also, thanks for the generosity of Agriculture and Agri-Food Canada in providing this study funding.

Finally, I would like to express my deepest gratitude for the constant support, understanding and love that I received from my husband Ahmad and my mother for her infinite and unwavering support also my father, my sisters and my brother specially my sister Forogh for always being there for me when I needed support during the past years.

Contents

List of Figures

List of Tables

Acronyms

CBN Cellulosic Biofuel Network

DAS Distributed Annotation System

GH Glycosyl Hydrolases

MSA Multiple Sequence Alignment

GFF General Feature Format

HMM Hidden Markov Models

SOA Service Oriented Architecture

SOAP Simple Object Access Protocol

REST Representational State Transfer

RelaxNG Regular Language for XML Next Generation

SGML Standard Generalized Markup Language

DTD Document type definition

EMBOSS European Molecular Biology Open Software

RDF Resource Description Framework

USA Uniform Sequence Address

DNA Deoxyribonucleic acid

RNA Ribonucleic acid

Chapter 1

Introduction

Biologists face two major challenges when performing experiments based on the biological data [LBW$^+$04]. First, the nature of the biological data is highly heterogeneous; i.e., different data types and formats are specified, each requiring the development of its own set of tools with which it can be manipulated. Also, these data may have been generated by isolated research groups around the world, who may have had their own standards and controls on the data. Second, the amount of available biological data has increased significantly in recent years as a result of the increased use of different computational tools and developments in computer hardware. Even though most of the databases and tools are available on the web, data mining and the integration of these tools is still a challenge. It should, however, be noted that the growth of the web has partially resulted in the growth of technologies that help people in the field of genomics. Accessing web services to analyze bioinformatics information is simple and requires no specific knowledge; however, it does require scientists to be expert enough to go through databases and web servers, and to skim the required data [ZCT09].

Biologists also need to be knowledgeable about different areas to accomplish their tasks and to deal with the problems that may arise, including [ZCT09, LBW$^+$04, SLDL06]:

- Challenges due to the complexity of multiple domains.
- Understanding the different data schemes and the programmatic interfaces they require.
- Tracking inconsistent jargon.
- Dealing with various data formats and frequent changes to the data structure.

These problems can be divided into two major groups: those related to data integration and those related to tool integration.

Numerous proposals have been suggested and developed in the literature to address problems with regards to simplifying access to genomic data and related tools, which will be explained in the following section.

1.1 Current Data and Tool Integration Solutions

Solutions regarding the data and tool integration problems can be divided into the three categories described below.

1. **Methods for data integration**: These methods include screen scraping, cross-referencing, data warehousing, view integration, web services, mashups and semantic web integration methods [GS08]. These methods have been developed to minimize the users' work, by having features such as automatics data mobility between programs. Also, these methods can simplify the data search process. As a result, users can find suitable data sets much easier. The data integration process involves tediously pipelining common touch-points, e.g., data values, names, identities, ontology terms, keywords, etc. In addition, different mechanisms have been employed in the above methods in order to proceed with

the data integration process. According to Goble and Stevens, these mechanisms include: “direct interlinking between database records, cross-database indexing, data exchange protocols, and the mapping of names and values between different data sets” [GS08]. For example, “SRS” [EUA96] and “Entrez” [SEOK96], are two samples of cross-referencing data integration systems. Both systems use a link integration mechanism which connects two entries in different data sources. “BioMART” [DMK+05] and “BioWarehouses” [LPW+06], are two other examples that use the data warehousing approach. This approach defines a central server in order to keep different data unified and updated. Among the data integration systems, the Distributed Annotation System (DAS), which defines a data exchange protocol, is one of the most popular data integration systems. The DAS protocol will be described in detail in Chapter 2.

2. **Desktop systems for tool integration**. Different desktop systems have been developed in order to give bioinformaticians the ability to access and organize various bioinformatics’ tools and also to automate complex data processing [NMM+09]. In this regard, workflow systems such as Taverna [MSRO+10] and Kepler [ABJ+04] provide access to distributed resources through the help of web service solutions such as BioMOBY [WL02] and SoapLab [SRO03].

3. **Web-based environments for tool integration**. Web-based environments such as the Biology Workbench [Sub98], PISE [Let01], Galaxy [GRH+05], GenePattern [RLG+06], BioManager [CA07], and Mobyle [NMM+09] have all been developed to provide services to biologists with minimum experience in computer science. In fact, they allow the user to access bioinformatics tools using a web interface rather than installing and maintaining the tools on their local servers. It should be noted that Mobyle is the most recent, more advanced version of PISE. The Mobyle portal will be described in detail in Chapter 2.

1.2 Motivation

The primary objective of the present thesis is to find a widespread and reliable data integration method that provides the available data to the bioinformatics community and also to integrate this method in an efficient environment that will allow users to use different bioinformatics tools.

For example, bioinformaticians in the Cellulosic Biofuel Network (CBN) Group, a large-size research group that is a collaboration between nine government laboratories and nine universities, have focused their research on the conversion of agricultural waste into biofuel. They have analyzed the fungal data to find a new way so that they can do the conversion in a faster and easier way. They have also had problems with regards to tool and data integration for their fungal data. They need a user-friendly tool integration environment that will accommodate a stable set of bioinformatics tools that they can use regularly as well as a data integration method that will allow them publish and use their data.

In order to select appropriate data and tool integration systems, the user requirements for such systems should be considered. In the following sections, the criteria for tool and data integration systems will be discussed, followed by a brief evaluation of the selected systems for the current study.

1.3 Criteria for Tool Integration Systems

The major expectations of biologists from a tool integration system are the following [LA04, NMM+09]:

1. The system should integrate a set of well-known and regularly used tools.

2. The system should simplify the management of analysis and comparisons of results.

3. The system should support the connection of tools in order to make scientists' tasks easier.

4. The system should be able to re-execute the previously used tools.

5. The system should support all of the above-mentioned features in an easy and understandable way.

6. End users should not be worried about adding and removing the tools.

Based on the above requirements, the tool integration system shall have the following five main criteria [NMM+09]:

1. **Usability:** As the end user in bioinformatics is usually an inexperienced user, usability is a very important issue for a tool integration system. Usability includes the following sub-criteria:

   (a) **Learnability:** The tool integration system shall be able to accommodate different tools and services with different interfaces. In order to minimize the amount of effort each user needs to put forth to learn how to work with the system, the system shall provide users with a similar interface for all of its tools or services.

   (b) **Clarity:** The system shall inform the users about the types of operations they can do, based on their data sets.

   (c) **Concept clearness:** The system shall contain the tools and services that have well-known names and functionality for the biologists.

   (d) **Helpfulness:** The system shall provide enough help and documentation to guide users in becoming familiar with the services provided by the tools.

2. **Domain specific functionality**: The ideal tool integration environment shall have the following capabilities [NMM+09, LA04, Sub98]:

(a) **Tool integration**: The main functionality of an integration environment is to allow tools to connect and work together. To do so, the following functions are essential:

   i. **Data type detection**: The system shall detect the data type of user data and the type of available tool or service outputs.

   ii. **Data type conversion**: The system shall be able to automatically convert the format of the data into a desirable format.

   iii. **Tool chaining**: The system shall support "tool chaining" in such a way that the output of one tool can be used as the input of another tool.

(b) **Environmental control**: The system shall allow a user to manage a research study in the system environment. The minimum requirements for achieving this goal are:

   i. **Bookmarking results**: The system shall allow a user to bookmark and save results for future use.

   ii. **Saving scenarios**: The system shall allow a user to save a designed scenario (e.g., tool sequences) in order to allow the tools in the scenario to run on a data set at any time.

   iii. **Tool execution history**: The system shall keep a list of tools that a user has executed in order to allow the user to re-execute them.

   iv. **Easy navigation**: The system shall allow a user to move easily between the results so that the user can compare them.

   v. **Browsing services or tools**: The system shall allow a user to browse a list of available tools or services.

   vi. **Service or tool search**: The system shall allow a user to search for available tools or services.

3. **Interoperability**: The system shall support communications with remote services [CRP05].

4. **Concurrency**: The system shall be able to execute tools concurrently in order to reduce the user's waiting time.

5. **Openness**: As the purpose of the system is to accommodate as many services as possible, adding a newly developed tool to the environment should be easy. As a result, the documents required that explain how to introduce a new tool or service to the system shall be clear. Also, the type of data that the system accepts shall be well-described [NMM+09].

1.4 The Platform Used for Tool Integration

Mobyle was selected in this thesis as a tool integration environment, as it provides functionalities that address most of the user requirements that were itemized in the Section 1.3. Some of the capabilities of Mobyle are listed as follows [NMM+09]:

- It provides biologists with a user-friendly environment in which users do not need to worry about installing tools and making updates (Criteria 1, Usability).

- Tool chaining is a simple task in Mobyle, with easy and automatic data conversion (Criteria 2.a, Tool integration).

- Mobyle can offer a history of tool executions and the result history, which allows users to have more control over their research (Criteria 2.b, Environmental control).

- Tools in the Mobyle portal work inter-connectively. Also, a Mobyle server can communicate with other Mobyle servers and web services such as BioMOBY web services (Criteria 3, Interoperability).

A more detailed evaluation of the Mobyle portal pertaining to these criteria will be described in Chapter 2, Section 2.3.1.2 .

1.5 Criteria for Data Integration Methods

In order to help biologists with publishing and integrating the requisite data a reliable data integration method is also required. The method should help scientists such as the CBN group and the bioinformatics community members to submit and collect data and annotations in a quick and easy way. For example, such a method would be needed to publish fungal data provided by the CBN group members. According to Dowell et al. [DJD+01], user requirements for such a method include the following:

- Independent research groups should have the ability to share the available data in different databases and formats.
- Each group should be able to update its data, with no need to ask a central authority to confirm and approve the data.
- Users should have access to the most recent data provided by each research group.
- There should be a protocol accepted by all research groups in order to unify the presentation format of the data and the communication method.

Based on such user requirements, the ideal data integration method should have the following criteria [DJD+01, PDK+07]:

1. **Non-central authority**: The method shall not require a central authority to approve the accuracy of the submitted data.
2. **Timeliness**: The method shall provide the most recent data to a user.

3. **Data exchange standard**: The method shall provide a standardized format for exchanging the data.

4. **Common identifier**: The method shall allow data providers to agree on a common identifier in order to relate the relevant data.

5. **Autonomy in creation of data structure**: The method shall allow a data provider to have enough freedom in terms of creating the format of a data source.

6. **Interoperability**: The method shall provide data in human-readable and machine-readable formats in order to interoperate with bioinformatics tools.

7. **Data retrieval**: The method shall allow a user to retrieve essential information in the available data sources.

1.6 The Method Used for Data Integration

The DAS protocol was selected in this thesis to help scientists publish data as per the following characteristics. These characteristics address most of the criteria regarding user requirements for the integration of data [DJD+01, PDK+07].

- The DAS protocol provides freedom to the data providers because of its decentralized authority (Criteria 1, Non-central authority).

- As a DAS server connects directly to the main database, any changes in the data will immediately affect the output of the DAS server (Criteria 2, Timeliness).

- The DAS protocol provides a coordinate system in order to support a common identifier (Criteria 4, Common identifier).

- The presentation format of the DAS data is independent of the structure of the data in the underlying database; in other words, the presentation of the DAS data follows a unified interface that works independently of the data structure (Criteria 5, Autonomy in creation of data structure and Criteria 3, Data exchange standard).

- The DAS protocol provides data in a machine-readable format, allowing it to work inter-operatively with other systems (Criteria 6, Interoperability).

- A DAS server registry allows users to find the available DAS servers, as well as the information required to communicate with such servers (Criteria 7, Data retrieval).

A more detailed evaluation of the DAS protocol pertaining to these criteria will be described in Chapter 2, Section 2.2.2.2.

1.7 Problem Statement

According to the problem pertaining to data and tool integration and after reviewing the available bioinformatics tools and databases, as well as data integration methods, Mobyle and DAS were found to be the best solutions for the purposes of tool and data integration. With this in mind, in this thesis two DAS reference servers have been developed for the DNA and protein sequence data that was provided by the CBN group (see Chapter 5).

However, no tool had been available to connect Mobyle to all of the available DAS reference servers, as well as the two previously mentioned DAS reference servers designed for the CBN group. For example, without such a tool, the Mobyle users have had to search for the URL address of the desired DAS reference servers and then query the servers one by one. Then, users have had to copy the results from several servers and paste the results into the Mobyle environment in order to use them for further analysis such, as multiple sequence alignments and phylogenetic studies. This process is a time consuming, confusing and an error-prone activity.

The solution of the problem was to design a new tool for the Mobyle portal, named “DasSeqFetcher”, that will be explained in Chapter 3.

1.8 Contributions

The main contribution of this research has been to develop a web-based software tool named “DasSeqFetcher”, in order to facilitate the transfer of data from DAS servers to the Mobyle portal. In fact, this tool was developed in order to combine the data retrieval abilities of the DAS protocol with the power of the Mobyle portal as a tool integration environment.

The DasSeqFetcher is a data aggregator tool that functions according to the DAS protocol. This tool meets the following user requirements:

- It provides the most up-to-date sorted list of available DAS reference servers.
- It can also connect to the tools that work under the Mobyle portal.
- By using this new tool, users do not need to collect data from different data sources; instead, the tool collects the required data from different DAS reference servers and imports them into the Mobyle portal.
- Users can select the server names from the tool’s server list and enter the sequence IDs as input parameters to the tool, and then the tool merges and returns the result sequences to the users in the FASTA format (a format for representing biological sequences).

The main user requirements for the newly developed DasSeqFetcher tool are listed as follow:

1. **list of server items**: The tool shall allow a user to access data from a wide range of DAS reference servers, and display the servers in a list box [MCP06]. The list box shall only show the operational reference server(s).

2. **Adaptability**: The tool shall automatically update its DAS server list based on new information by connecting to the DAS server registry, with no need to run a separate program.

3. **Merging the results**: The tool shall merge the fetched sequences from several servers into one output.

4. **Concurrency**: The tool shall allow a user to simultaneously query more than one DAS reference server for a list of sequence IDs.

5. **Timeliness**: The tool shall return the most recent data pertaining to the sequence ID(s) that have been entered by the user.

6. **Robustness**: The tool shall not allow a user to enter incorrect data. Also, it shall check the functionality of the DAS reference servers that users are trying to connect with, in order to fetch the required data. Moreover, the tool shall query alternative DAS reference servers that users have selected, in case the first data source does not provide the data. Also, it shall inform the user about unavailable sequences.

In fact, adding the DasSeqFetcher tool to Mobyle improves the ability of the portal to access to more data sources as most of the tools installed in Mobyle need to be fed by data, such as protein or nucleotide sequences. Moreover, there are currently many bioinformatics databases that provide their data via a DAS reference server. At the time of this thesis, there were around one hundred and fifty available DAS reference servers. Consequently, DasSeqFetcher gives the Mobyle a greater flexibility by connecting it to a large number of DAS reference servers.

Another contribution of this work is the development of two DAS reference servers for retrieving the DNA and protein sequence data that are kept in “mycoCLAP” [MWP^{+}10], a database of characterized lignocellulose-active proteins. This database was provided by the CBN group. These two servers will provide the mycoCLAP data based on the DAS protocol.

1.9 Thesis Outline

The rest of the thesis is organized as follows: Chapter 2 briefly reviews the descriptions of data integration and tool integration solutions, including the DAS protocol and the Mobyle portal. Chapter 3 details the DasSeqFetcher specification, architecture and design. A review of the technical issues relevant to the DasSeqFetcher is presented in Chapter 4, where an evaluation of the DasSeqFetcher program is made, followed by a comparison with related works.

Chapter 5 provides specific details about the specification, design, and implementation of DAS DNA/protein reference servers pertaining to the data that belongs to the CBN group. Chapter 6 presents a case study to show how the inter-operation of the DasSeqFetcher tool and the Mobyle portal can help biologists complete a study. Chapter 6 also shows the power of the DasSeqFetcher to retrieve sequence data from a DAS reference server during a phylogenetic tree study. It is followed by conclusion and discussion on future work in Chapter 7.

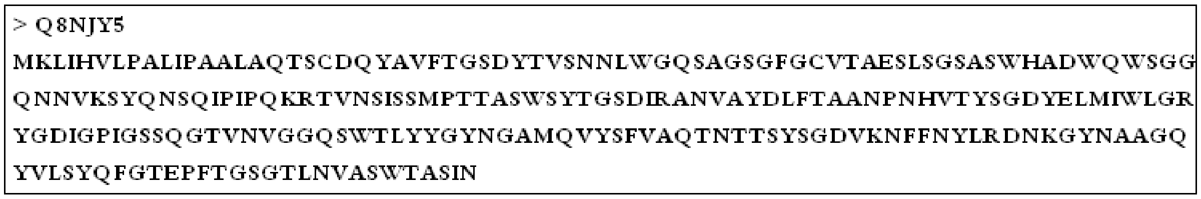
Chapter 2

Bioinformatics Tools and Background

This chapter gives a brief introduction to proteins and protein sequences, as they are important data for the current work. Also, two data integration methods, distributed annotation system (DAS) and web services are explained in "Data Integration Solutions" section. In addition, three tool integration systems, Mobyle, BioMoby, and Taverna are described and compared in "Tool Integration Solutions" section.

2.1 Definition of Protein and Protein Sequences

Proteins are the vital components of life. They perform many important functions for living organisms and play a key role in cellular processes. Proteins are sequences of amino acids that are folded into a globular structure. There are four levels of protein structures: primary, secondary, tertiary and quaternary structures. The primary structure is the sequence of amino acids that makes up the protein.The arrangement of amino acids in the primary structure of a protein is based on the gene (DNA sequence) that, in turn, is transcribed and translated to produce the protein [PR04]. All of the known protein sequences have an ID on the well-known protein databases such as “UniProt”. For example, “Q8NJY5” is a protein ID that is provided by UniProt database, as shown in Figure 1.



```
> Q8NJY5
MKLIHVLPALIPAALAQTSCDQYAVFTGSDYTVSNNLWGQSAGSGFGCVTAESLSGSASWHADWQWSGG
QNNVKSYQNSQIPIPQKRTVNSISSMPTTASWSYTGSDIRANVAYDLFTAANPNHVTYSGDYELMIWLGR
YGDIGPIGSSQGTVNVGGQSWTLYYGYNGAMQVYSFVAQTNTTSYSGDVKNFFNYLRDNKGYNAAGQ
YVLSYQFGTEPFTGSGTLNVASWTASIN
```

Figure 1: The sequence for “Q8NJY5” protein ID

Proteins with specific functions have domains that are responsible for those functions. A domain is part of a protein polypeptide sequence. The domains are independently stable and folded [CGVT03]. An example of two domains for a “Bacteria” protein with sequence ID “C3Q4G7” is shown in Figure 2. According to the Pfam [BBC+02] database, the protein includes two “Glycogen hydrolyzes” domains: “Glyco_hydro_43” and “Glyco_hydro_28”.

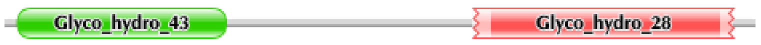


Figure 2: Pfam database “Accession ID search”: result for protein ID “C3Q4G7”

In the case of the CBN project, the investigation is currently focused on producing ethanol from biomass materials such as crop residues. This conversion can be carried out in different stages. In one of the stages, an enzyme (protein) family called “Glycosyl Hydrolases” hydrolyzes cellulose and hemicellulose into small sugars. The sugars are transformed to ethanol in a fermentation process [HAJvRH03].This process is shown in Figure 3.

The researchers in the CBN group require a suitable method to gather the target sequences that are scattered over different data sources. The researchers are also interested to have a convenient environment to integrate the tools that they use frequently. The next sections explain the data and tool integration methods that are reviewed for these purposes.

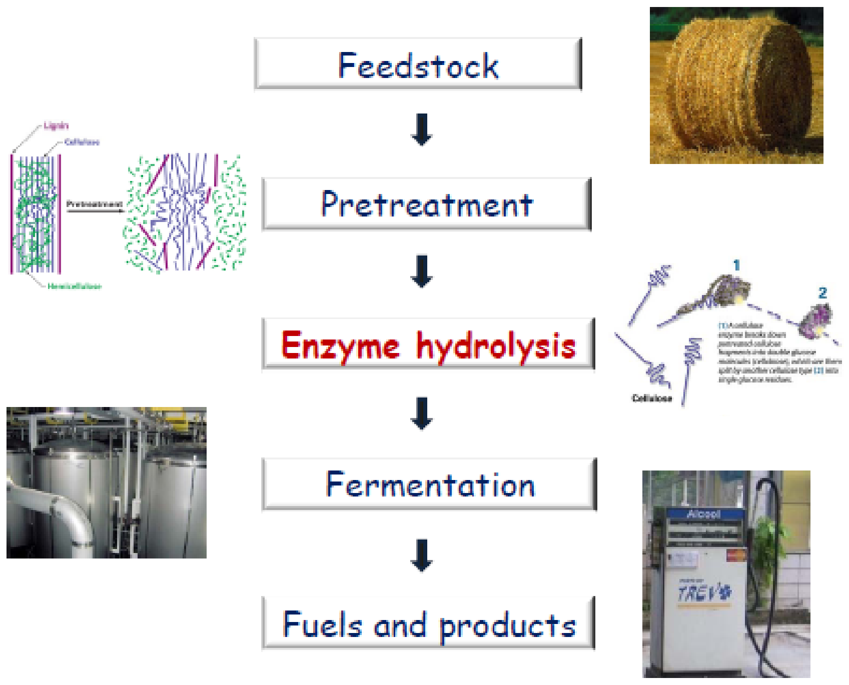


Figure 3: Conversion of lignocellulose to fuels [Tsa09]

2.2 Data Integration Solutions

This section describes the web service and the DAS protocol solutions for data integration problem in bioinformatics. Also, DAS is evaluated against the data integration criteria explained in Chapter 1, Section 1.5.

2.2.1 Web Service

In recent years, web services have become a hot topic in bioinformatics. Many research groups have taken advantage of web service abilities in order to publish their data and services for the bioinformatics community. On the other hand, there are other technologies that can integrate data, such as the well-known DAS protocol, which can be used extensively to publish biological data. Web services and DAS protocols have some similarities and differences, which will be further discussed.

DAS and web services follow a similar architecture. Web services work under a Service Oriented Architecture (SOA), made of three main components: the service provider, the service registry and the service consumer. The DAS protocol also follows the same structure, i.e., there is a DAS registry by which all DAS references and annotation servers are registered. There are some institutions that provide DAS servers and there are also various programs and tools (clients) that take advantage of the services provided by DAS servers [JAB+08].

DAS is comparable to web services in terms of the content of their communication methods, even though the methods are not entirely similar. Web services work under the Simple Object Access Protocol (SOAP) and provide a response by encapsulating it in an XML envelope. The XML responses do not follow a defined format. It is up to a service developer to decide which XML elements will be include in the response. On the other hand, to use a DAS service, clients send an XML message to the service provider to request the service [PDK+07]. According to Carole Goble et al. [GS08], considering the above-mentioned technique of request and response, and also considering inadequate service definition and documentation of web services, SOAP is a heavy protocol with insufficient features for data integration. However, it may work well in other contexts. For example, SOAP services can be employed to do heavy calculations on a set of data. As a result, research groups have typically preferred to provide their bioinformatics tools as a web service.

On the other hand, the DAS protocol follows the Representational State Transfer (REST) model, which is a lighter protocol than the SOAP. Consequently, DAS clients use a well-defined URL to query DAS servers, and instead of making a heavy XML message for the response, the response is placed in an XML file. The response file should be made based on the DAS protocol response format, with specifications that will need to be accepted by clients as well as by the DAS registry, as a valid DAS server response [PDK+07]. These rules make the DAS a more powerful tool in biological data integration.

Another contrast between DAS and SOAP web service is that the type of data that they provide is different. DAS data are usually pre-calculated or need very little computation, whereas the data requested from SOAP services need heavy calculations most of the time [PDK$^+$07].

Based on reviewing the available bioinformatics data integration methods, the DAS protocol was selected as a data integration method for this thesis. The next section describes the DAS protocol, an evaluation of the protocol against the data integration criteria and the available libraries for implementing the protocol.

2.2.2 Distributed Annotation System (DAS)

DAS is a protocol that was developed to integrate data from heterogeneous databases [DJD$^+$01]. In brief, the DAS protocol offers a standard data representation format, a communication method, and a query format for exchanging biological sequences as well as their annotations. In fact, a DAS server is separate from the database architecture which stores the data [FSJ$^+$07]. Each research group can have separate control over their own data. In other words, by using DAS, it is possible to integrate annotation data in a decentralized way. These data then can be used by a DAS client software [DJD$^+$01].

2.2.2.1 DAS Protocol

The DAS protocol is a simple HTTP-based client-server system that defines the method of data exchange [JAB$^+$08]. The client software sends a request to a DAS server in a URL format, similar to the one shown in Figure 4. This request, which is called a “query”, contains the DAS command that specifies the information that the client is requesting from the server. The server response is a “sequence” or an “annotation” data that is returned to the client in the XML format. The XML specification defines which response is acceptable.

In this process, the client software plays an important role. Servers provide the

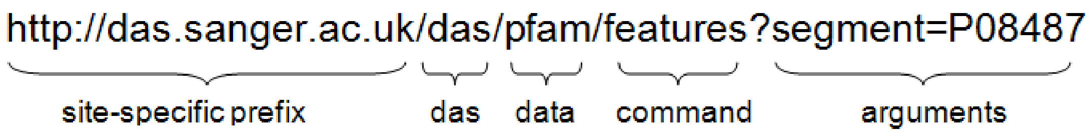


Figure 4: DAS query sample [DJD+01]

data that has been standardized by using the DAS protocol, regardless of how clients intend to use them. Therefore, the client software is the one that collects all related annotations [PDK+07]. The five main components of the DAS protocol architecture include:

- Coordinate systems
- Reference servers
- Annotation servers
- Server registry
- Clients

All of these components will be briefly explained, below.

Coordinate Systems

In the process of integration of data, clients need to find all of the DAS annotation servers that can annotate a particular reference object. A reference object is a biological sequence that has a unique and stable identifier such as a protein or a DNA sequence [PDK+07]. The identifier is used by all clients and servers to connect every piece of related data that has been scattered over different servers. The description of a set of reference objects is called "coordinate systems" or "namespace". An example of a coordinate system is a set of protein sequences belonging to a specific species. It should be noted that if a reference object has several newer versions, it needs to have a "version tag" in order to be distinguishable by all systems. This is a result of the fact that each version may have nonequivalent annotations. The coordinate systems are described by using the following information [LNE06, PDK+07]:

1. Type of annotatable entity, which could be a chromosome, gene, protein sequence or protein structure.
2. The authority that defines the coordinate system. This can be the name of an institution such as UniProt or Ensembl.
3. The version of a set of sequences.
4. The organism that the data comes from.

Reference Servers

A reference server provides the reference sequences. These sequences are the core pieces of data to be described with annotations. In fact, a reference server presents one or more of the DNA, protein, and RNA sequences, and the reference sequence map. A map refers to a number of "entry points" that facilitates access to the data by the client. An entry point can be a complete sequence, such as a chromosome, or just part of the sequence that is called a "substructure" [DJD+01]. In addition, since a map has a hierarchical structure, in a case where the client provides a part of the map, the server can return the parent (e.g., the sequence to which this part belongs to) and children (e.g., the subsequences which belong to this part) of that entry point.

Annotation Servers

Annotation servers are responsible for providing a list of annotations that describe different parts of a sequence. These annotations can be non-positional or positional. Non-positional annotation refers to the information about the whole sequence, not just a specific location, such as a list of the literature available for a sequence. On the other hand, positional annotation is the part of the data that is about a specific position on a sequence, such as a specific domain on a protein sequence. Each of these annotations consists of the following information: the border of the described region based on the entry points, an accession number which refers to the server provider, the category that the annotations belong to and a style sheet that explains how the client software should show the annotations to the user [DJD+01].

It should be noted that all annotations that are provided by a DAS annotation server are pre-calculated.

Server Registry

With the increase in the use of DAS servers by many research groups, it is hard for clients to find new servers and the type of service offered by them. As a result, a useful service called the "DAS registry" was developed. This registry keeps track of the new servers and the existing DAS servers' statuses [JAB+08, PDK+07, LNE06]. To take advantage of this service, a "sources command" has been added to the DAS protocol. This command offers useful information about the capabilities of DAS servers for the clients, such as: the URL addresses of servers to help clients communicate with the server; a short description about the type of data that the server provides; and a machine readable metadata that helps the client software find out whether the server provides the clients with the relevant data they need [PDK+07].

The responsibilities of the DAS registry are described as follows:

1. It provides a web page with all available servers and their capabilities. Also, clients can have a list of available servers in the form of XML. The XML can be used within the DAS clients programs.

2. It periodically checks all the registered servers to ensure that they are properly functional and it also validates them to make sure that the response in XML format follows the DAS protocol.

3. It can categorize DAS servers based on the coordinate systems of their data. This categorization helps the users to find the target servers more easily [PDK+07].

Clients

A DAS client can use and integrate the data provided by one or more DAS servers. The client can also connect to a DAS registry in order to obtain information about the DAS servers [DJD+01].

Some famous DAS clients include the following: Dasty, Proview and Jalview. The latter is a multiple sequence alignment viewer that utilizes the DAS protocol to provide a more powerful environment for users [PDK+07, JVD+05, Ola05].

DAS Architecture

As shown in Figure 5, below, at the beginning, the client connects to the DAS registry to get the list of available DAS reference and annotation servers, shown below as number "1". Then, the client sends a query to the reference server, using its URL address to request a sequence, shown below as number "2". The server then puts the response in XML format and sends it back to the client, as in number "2". Having the reference sequence information, the client then searches the registry for annotation servers which provide all relevant annotations, as in number "1". Then, the client fetches annotations by querying these servers, as in number "3". It is the client's responsibility to find all corresponding annotations and to compose and display them in a suitable format [PDK+07].

2.2.2.2 DAS Evaluation

The DAS fulfills most of the requirements regarding data integration for bioinformaticians as explained in Section 1.5. In this section an explanation is provided regarding how the DAS can support the criteria for data integration. The seven criteria are described below [DJD+01, PDK+07]:

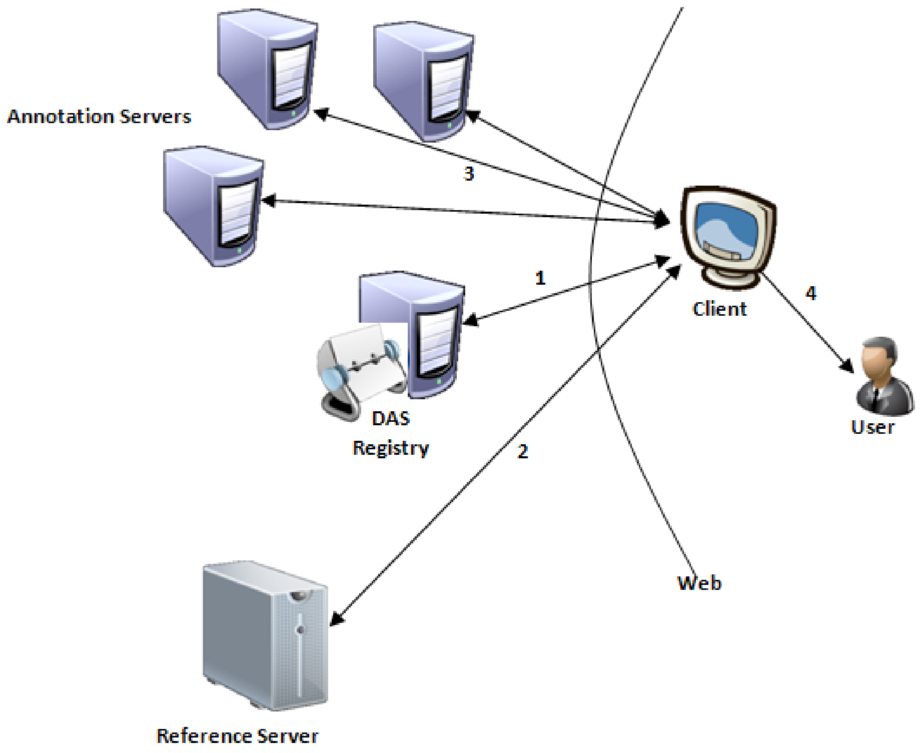


Figure 5: DAS architecture [Dow01]

1. **Non-central authority**: The DAS protocol completely meets the decentralized authority requirement. Each research group has its own data and its own DAS server. Unlike some approaches to integrate data such as having a data warehouse, there is no need to have a center to control the correctness of data or to aggregate the other research groups' data [DJD+01, Bio10].

2. **Timeliness**: As a DAS server connects directly to the main database, any changes in the data will immediately affect the output of the DAS server. Therefore, the clients can have the most recent data in response to their requests, proving that the DAS meets the criteria for timeliness.

3. **Data exchange standard**: The DAS protocol provides a very organized communication model. It uses a specific URL request to query data from the reference and annotation servers. The response is in XML format and is based

on the DAS protocol specification. The protocol assures that all DAS servers behave the same way for every data request and for every produced response.

4. **Common identifier**: The DAS protocol provides a coordinate system in order to have a common identifier. All of the servers and clients use the same identifier to recognize a piece of data.

5. **Autonomy in data structure**: Regarding "autonomy in the data structure", research groups provide their DAS servers alongside their official databases [FSJ+07]. If the underlying database structure is changed, then the clients are protected against this modification and can have access to the data without any adjustment. It is because of that the DAS has a specific protocol to show the data to the end user. In this situation, it is the research group's responsibility to modify its DAS server to match the server with the new data structure.

6. **Interoperability**: To support interoperability with some bioinformatics tools and services, all of the information provided by DAS servers is machine-readable. Also, the DAS registry provides clients with information that allows them to verify which server offers the information they are seeking [DJD+01, Bio10].

7. **Data retrieval**: The DAS protocol possesses a registry that keeps all required information about DAS servers, such as their capabilities and URL addresses. Users can either search the registry manually to find a specific DAS reference server or can connect to the registry with a program that automatically reads the entire data. The registry periodically checks all registered servers to ensure that they are functioning properly, and it also validates them to make sure that their response in XML format follows the DAS protocol.

2.2.2.3 Implementation of DAS Protocol

Section 2.2.2.1 explained the goal and architecture of the DAS protocol. In order to implement the protocol, it is necessary to use some tools that embody the protocol in the real world. These tools should be easy to use and should allow research groups to develop their own DAS servers without worrying about the XML format and the web development process. Several code libraries have been created to fulfill this task, both for the DAS client and the server. Some of them have been written in Java (e.g., Dazzle, MyDAS and Dasobert) and others in Perl (e.g., Proserver and LDAS), each of which has their own advantages and disadvantages. A server library, named "Dazzle", and a client library, named "Dasobert", are illustrated as below.

Dazzle

Dazzle is a Java implementation for a DAS server. It was written as a servlet and uses BioJava APIs to communicate with clients. Dazzle is a modular system that can support more than one data source at a time. It contains several different plug-ins, which can help in the creation of protein and DNA references and annotation servers. These plug-ins can read data from a simple text file, such as a GFF data file or they can connect to a database. These plug-ins can first read the data, and secondly they can convert this data to the DAS XML format [DJD$^+$01, Dow].

Dasobert

Similar to the server library, there are some APIs written in both Java and Perl languages for developing DAS clients. Two examples of such APIs are "Bio::Das::Lite" and "Dasobert". Dasobert [Prl07], based on BioJava APIs, can connect to the DAS registry and DAS reference and annotation servers. Dasobert currently supports DAS/1 and some aspects of DAS/2.

2.3 Tool Integration Solutions

In this section, some of the most attractive and "state of the art" methods of tool integration in bioinformatics are introduced: Mobyle, Taverna and BioMoby. Also, the two latter systems as service integration methods, are compared to the Mobyle portal, as a portal integration method. It is worth noting that even though these systems use different methods of integrating tools and services, they all have some similar functions that will be compared.

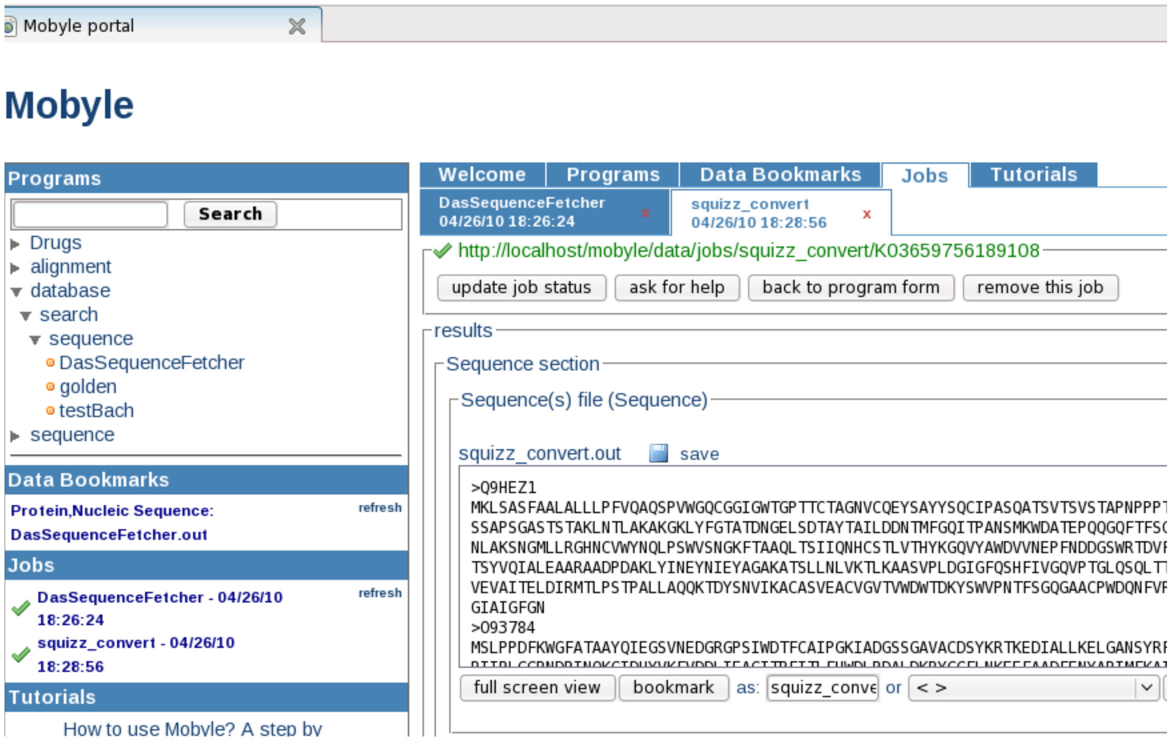
2.3.1 Mobyle

Mobyle is a web-based portal that integrates a large number of bioinformatics tools. At the beginning stages in design of this system, several interviews and workshops were established to help system designers fully understand the needs of biologists. As a result, although this system uses advanced technologies such as web services, remote software execution and a dataflow model, it provides an easy-to-use, reliable environment for managing data and for linking tools [NMM+09].

In this section we will discuss the architecture of Mobyle, followed by the evaluation of the portal against the tool integration systems criteria. Also, the Mobyle data aggregator tools are introduced at the end of the section.

2.3.1.1 Mobyle Architecture

Mobyle provides bioinformatics users with many capabilities to facilitate their research. For example, the main web page of the Mobyle portal contains two panels, as shown in Figure 6. The left panel keeps a list of available tools that have been installed in the current web portal. The search facilities in this panel allow users search the tool list by inputting a tool name. In addition, a list of remote Mobyle servers that are connected to the current portal is shown in the panel. Moreover, users can see a list of bookmarked data as well as a list of jobs that have already been executed in the left panel.



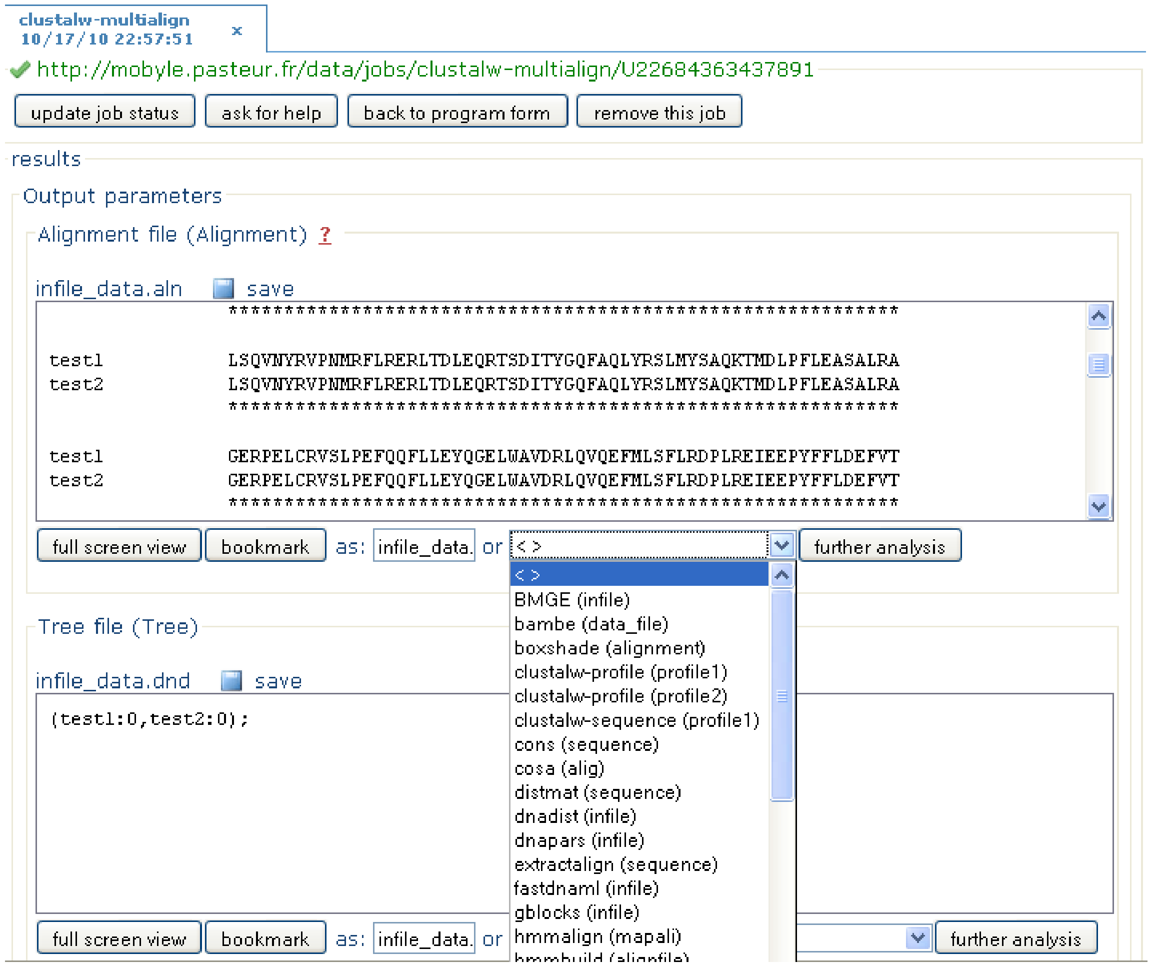
http://localhost/mobyle/data/jobs/squizz_convert/K03659756189108

```
>Q9HEZ1
MKLSASFAALALLLPFVQAQSPVWGQCGGIGWTGPTTCTAGNVCQEYSAYYSQCIPASQATSVTSVSTAPNPPP
SSAPSGASTSTAKLNTLAKAKGKLYFGTATDNGELSDTAYTAILDDNTMFGQITPANSMKWDATEPQQGQFTFS
NLAKSNGMLLRGHNCVWYNQLPSWVSNGKFTAAQLTSIIQNHCSTLVTHYKGQVYAWDVVNEPFNDDGSWRTDV
TSYVQIALEAARAADPDAKLYINEYNIEYAGAKATSLLNLVKTLKAASVPLDGIGFQSHFIVGQVPTGLQSQLT
VEVAITELDIRMTLPSTPALLAQQKTDYSNVIKACASVEACVGVTVWDWTDKYSWVPNTFSGQGAACPWDQNFV
GIAIGFGN
>O93784
MSLPPDFKWGFATAAYQIEGSVNEDGRGPSIWDTFCAIPGKIADGSSGAVACDSYKRTKEDIALLKELGANSYR
```

Figure 6: Mobyle portal interface

The right panel contains a main window that may show the interface of a tool, a result of a job or a list of bookmarked data. In addition, in the panel, there is a horizontal bar containing three main tabs: programs, data bookmarks and jobs. Using these tabs, users can easily switch between the executed tool interface, the bookmarked data and the results.

In the Mobyle environment, users can chain programs by sending the result of one tool as the input to another tool. If the output data format of a tool is not compatible with the input of the other, Mobyle can automatically convert it to a compatible format. Furthermore, Mobyle offers a list of tools that can work on the results of a tool, and these tools are shown in a list box [NMM+09]. Figure 7 shows the results of the “Clustalw” tool and the list box that is located close to the “further analysis" button. The list box contains a list of tools that can work with the Clustalw results.



```
                ************************************************************


test1           LSQVNYRVPNMRFLRERLTDLEQRTSDITYGQFAQLYRSLMYSAQKTMDLPFLEASALRA
test2           LSQVNYRVPNMRFLRERLTDLEQRTSDITYGQFAQLYRSLMYSAQKTMDLPFLEASALRA
                ************************************************************


test1           GERPELCRVSLPEFQQFLLEYQGELWAVDRLQVQEFMLSFLRDPLREIEEPYFFLDEFVT
test2           GERPELCRVSLPEFQQFLLEYQGELWAVDRLQVQEFMLSFLRDPLREIEEPYFFLDEFVT
                ************************************************************
```

```
(test1:0,test2:0);
```

Figure 7: Proposed tools for further analysis in Mobyle

As previously mentioned, the primary role of Mobyle is in its ability to chain tools. Mobyle solves the data type incompatibility problem by using a multidimensional data type. There are four categories of data that are supported by Mobyle [NMM+09]:

1. The "biotype" category is used to present the biological data, such as DNA and protein.

2. The "datatype" category describes the structure of the data, i.e., sequence, alignment or matrix.

3. The “basic type” category, refers to non-biological data, such as a string and an integer.

4. The “format” category describes the data format of bioinformatics tools, such as a FASTA sequence or a “phylip” distance matrix.

On the other hand, Mobyle asks tools to select their input and output data types among the four provided categories of data types. If a new data type needs to be added to the system, the tool developer should develop a new class that will implement the “DataType” class, directly or indirectly. Using the data categories defined above, Mobyle can provide the following services [NMM+09]:

1. It identifies and verifies the format of a user data and changes this format to the type that a user needs by employing convertor tools available in the portal.

2. It selects a set of programs that can accept the current results as their input.

As a server, Mobyle can chain local tools as well as connect to other Mobyle servers. A Mobyle server can send the user request to another Mobyle server to process the request remotely by using the Mobyle Network protocol. When the result is ready, the remote server returns the result to the initial server and the user can continue working with it, similar to the situation in which the user works with a local tool. This architecture is shown in Figure 8.

It should be mentioned that the programs that will be installed in Mobyle should be in an executable format and can be called from the command line.

RelaxNG

In this section, the RelaxNG XML schema format is explained. Mobyle uses this format to describe the XML files. More specifically the XMl file that describes a tool interface for Mobyle, is written in this format.

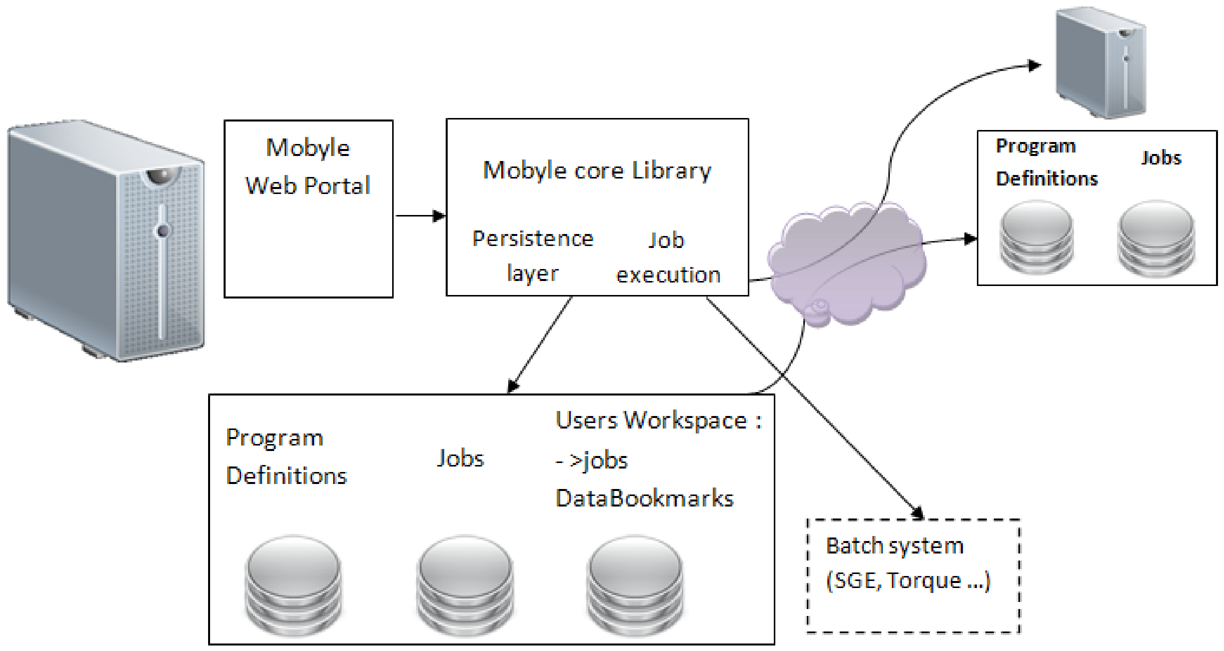


Figure 8: Mobyle network architecture [NMM+09]

RelaxNG (Regular Language for XML Next Generation) is defined as an XML Schema language. RelaxNG is a language that is used to describe XML files. Schema languages, in turn, help keep an XML file in a specific format by adding constraints to the structure and content of an XML document. The RelaxNG Schema file comes in an XML format, even though a compact version of the file can be used as well. Although RelaxNG is simpler to use compared to another schema language such as "XML Schema", it has not yet become very popular in the programming community [VdV04].

To integrate a new tool with Mobyle, a tool developer can specify the name, type and format of all arguments, using the Mobyle RelaxNG format. Perl and python scripts can be used in this XML file to describe the dependency between arguments and to explain how they should be placed in the command line. This XML file will then be used to show the program interface [WUP09].

Some important characteristics of RelaxNG are described as follows [VdV04]:

1. It is simple and easy to work with.

2. It can be provided in two different formats: XML and compact.

3. It supports unordered and mixed content.

4. It has the ability to work with XML namespaces.

It should be added that RelaxNG has been designed based on XML DTDs. As a result, it is easy for those who already have experience with XML DTDs and SGML (Standard Generalized Markup Language) to learn and to work with the RelaxNG pattern. In addition, as the XML DTDs have been used for a long time and their problems have been solved over time, RelaxNG is more reliable than other formats [VdV04].

2.3.1.2 Mobyle Evaluation

In this section, the Mobyle system will be evaluated according the six criteria that were explained in Section 1.3.

1. **Usability**: As user satisfaction is the most important requirement for a tool integration environment, Mobyle portal developers conducted an elaborate study on the needs of biologists, published by Letondal et al. [LA04]. To evaluate the usability of the system, a test session was held with the participation of 16 users. The results were such that although some changes were needed to make the system more clear, the system worked well and was not complicated for inexperienced users [NMM$^+$09].

   The four usability parameters can be defined as follows:

   (a) **Learnability**: Based on the results of the aforementioned study, in terms of usability, Mobyle provides a unified interface for all the tools that are installed in its environment. It makes the learning process simple for the end user, since all programs generally have the same configuration for execution, importing data and moving the results produced to the next tool in a chain. Also, the format of the result windows and error files is identical for all the Mobyle tools.

(b) **Clarity**: To satisfy the requirement for clarity, Mobyle automatically suggests a list of tools that can perform further analyses on the results of a tool. The tools have been suggested since their input data format is compatible with the results data format produced by previous tool. The user can easily select the relevant tool from a smaller, but more accurate list of tools. In addition, coming up with a scenario in Mobyle is not complicated, making a new chain of tools does not require a lot of training or effort.

(c) **Concept clearness**: Most of the tools that are installed in a Mobyle environment are well-known and biologists are familiar with them. Also, each research group can install its own new tool under their local Mobyle portal. A Mobyle administrator can also specify a commonly used name for tools that otherwise have unfamiliar names.

(d) **Helpfulness**: Mobyle asks tool developers to provide a help document as well as examples of their tools.

2. **Tool integration**: Mobyle supports tool integration by providing automatic data type detection and conversion. This eases the job of users, since they do not need to worry about finding the correct tool or service to convert the type of data. In addition, some convertor tools are provided in case a user wants to manually change the data format. Based on these functionalities, a Mobyle portal allows a user to chain the tools.

3. **Environment control**: By presenting an execution history of tools, as well as data bookmarking and giving a result history, Mobyle gives the users a range of control over the environment. Mobyle provides these services by creating a

personal workspace for each user and by keeping information on user accounts, even though Mobyle is also accessible to unregistered users or "guests".

Mobyle offers a tool navigation tree and a search service to allow users to find the appropriate tool in the list of available tools in the portal. These tools could be scattered over several Mobyle servers.

4. **Interoperability**: A Mobyle portal can connect to remote servers and can use their services in two different ways:

   (a) By connecting the portal to other available Mobyle servers and by sending a user request to the servers to remotely process the request.

   (b) By accommodating web services, especially, BioMoby web services by using the "PlayMoby" tool.

   In both cases, once a result is ready, the remote server will return it to the initial server. This result will also be shown in the initial server. This result is similar to the local tool's execution result [NMM$^+$09].

5. **Concurrency**: In the Mobyle environment, tools can work concurrently on different datasets. However, a tool cannot start working by using a part of a result from the upper stream tool in a chain.

6. **Openness**: Regarding the criteria of openness, adding a new tool to a Mobyle portal is quite easy when it has a simple interface. Each tool needs an XML file to describe its input and output parameters, and an executable file of the tool to be installed in Mobyle. The system administrator installs these tools on the Mobyle server, thus end users do not need to worry about the installation of the tools. However, if a new tool with a complicated interface (e.g., a tool has a dynamic list box) needs to be developed, the design process would be challenging, as generally there is no elaborated document for developing such

tools in Mobyle. As a result, developers will need to find previously designed tools to have an idea about how to design a new tool interface.

2.3.1.3 Mobyle Data Aggregation Tools

There are two tools, namely “Golden” and “seqretFromUSA”, available in Mobyle for users to fetch the required sequences from bioinformatics data sources. These tools are explained briefly, as follows:

Golden

“Golden” is a program that retrieves entries from indexed flat files. It can provide sequences from nine databases, such as “Genebank”, “Prosite” and “UniProt”. Users can search these databases to find a sequence by inputting a sequence entry name or an accession number.

The number of databases that can be accessed by Golden is static, and a server administrator is needed to change the Golden setup if any new database is added to the system.

Golden can return one sequence from one server at a time, which puts a limit on the number of programs that can use its output. For example, some programs, such as “Multiple Sequence Alignment” (MSA), need more than one sequence to function so that they cannot use Golden to import the required sequences.

SeqretFromUSA

“Seqret” is a program from “EMBOSS” (European Molecular Biology Open Software Suite) that can read sequences in different formats from different sources, such as databases, files and the command line. Seqret can return or write these sequences in many different places, such as on a screen or in a file [RLB$^+$00]. “SeqretFromUSA” is a version of Seqret program that concentrates on retrieving data from databases.

Users can enter a list of sequence IDs to fetch these sequences from different databases. Each entry consists of the abbreviation of the database from which they want to fetch the sequence, followed by a colon ":", and the sequence ID. Contrary to "Golden", this program can return more than one sequence from different data sources at the same time. But, it is not easy for users to work with SeqretFromUSA, as they should know the abbreviation of each database. Moreover, the program needs an administrator to keep its data up-to-date.

As these tools have some limitations, a program, namely DasSeqFetcher, was developed and integrated with Mobyle portal that will be compared with the above tools in Chapter 4, Section 4.3.

2.3.2 BioMoby

BioMoby is a bioinformatics web server registry which allows biological data sources and analytical tools and services to work together, using standard ontologies to annotate services [WL02].

BioMoby uses ontologies to define not only the biological intent and/or semantics of the data, but also to define the syntax of the data [LBW+04, GW06]. BioMoby aims to provide an architecture that facilitates biological data source communication, using the following conditions [WGFS03]:

- Exchanging data using a common format.
- Providing machine-readable information to introduce the data context and service for the clients and servers.
- Providing automated and manual data and service discovery.

The BioMoby project has already been subjected to two major architectures: "MOBY-S" and "S-MOBY". MOBY-Services or "MOBY-S" (web services oriented) services are simplified services as compared to WSDL (Web Services Description Language) and they also have single operations, inputs and outputs [WL02].

Additionally, Semantic-MOBY or S-MOBY (semantic web oriented) services extensively use semantic web technology that is based on the Representational State Transfer (REST) [Fie00] architectural style. S-MOBY was first used in 2001 by Damian Gessler at the "Model Organism Bring Your Own Database Interface Conference (MOBY-DIC)", held in Emma Lake, Saskatchewan [WL02]. The idea was to standardize the exchange of information, and to make access to analytical resources easier and more effective. In continue, S-MOBY and MOBY-S will be briefly explained.

The four main components of Semantic-MOBY are described as follows [LBW+04]:

- *Service interfaces* are simply web resources that can be reached by well-known protocols, such as HTTP and FTP.
- *Semantic descriptions* are presented in OWL-DL (Web Ontology Language-Description Logic) and are matched with a superior ontology.
- *Domain ontologies* are developed and are published on the web by the bioinformatics society.
- *Semantic search engines* are provided . Service providers can publish the addresses and semantic descriptions of their services that can be found by web crawlers.

On the other hand, MOBY-S follows a web service architecture. This means that data sources and bioinformatics services publish their interfaces in a public registry. Then, clients query the central registry to find the service that they are interested in and they can then call up the service in order to complete their jobs.

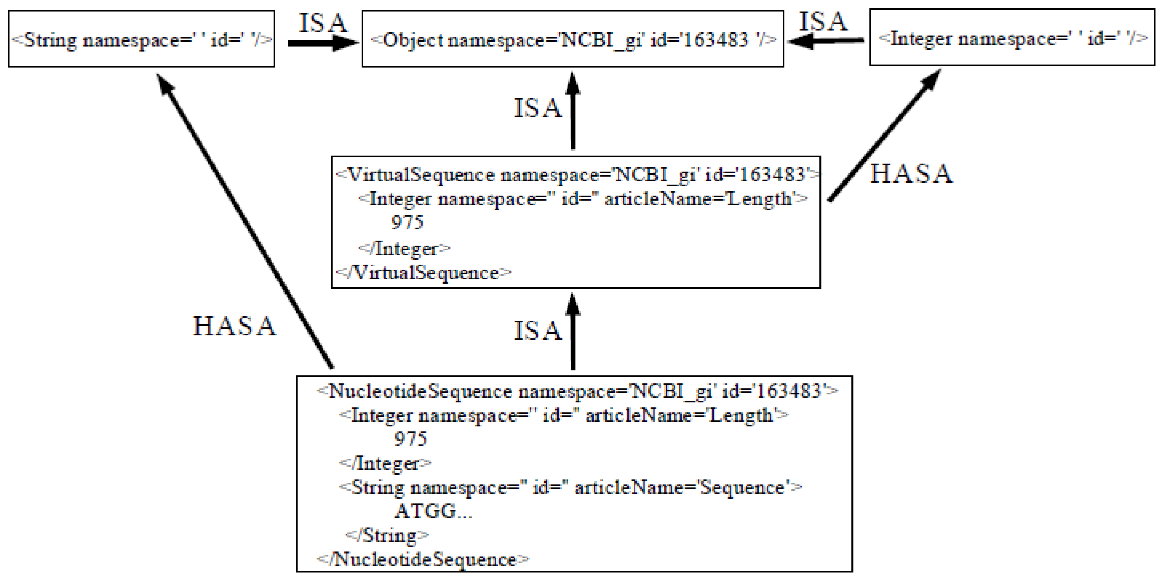
However, unlike the regular web services, MOBY-S uses ontologies to describe data and services and to evaluate how a service can act on a piece of data. Also, it utilizes the semantic web to improve the discovery of its services.

There are two main groups of ontologies in MOBY-S [WGFS03]: the class ontology, which describes the data types, and the service ontology, which manages the transformation type elements of the service signature. The service signature explains the service input parameters and the data type of the service output. In the class ontology, each piece of data, called "Object", is described as a class and has to directly or indirectly be inherited from the main object class.

As shown in Figure 9 below, regarding class ontology, all classes have two mandatory attributes, named the ID and the namespace. These attributes provide enough information in order to uniquely identify an object. All name spaces are selected from a list of specific terms. In addition, the optional "articleName" attribute can be used to explain the meaning of an object to the user whenever needed. For example, in Figure 9, in the Nucleotide Sequence class, the second node is a string that holds the DNA sequence. This node shows that the articleName is equal to the sequence, which explains the meaning of the string kept in that class.

There are two types of relationships between classes and instances of classes: ISA and HASA. ISA defines a parent-child relationship, which means that the subclass (child) has the same attributes of the super (father) class. It helps services to work with a greater variety of data types, rather than the specific data type that they introduced in their signatures. The reason is that a service can work with a superclass data type whenever the required data type class is not provided.

There are some primitive objects introduced in MOBY-S, such as string and integer. Primitive classes are the children of the main Object class. They are the only classes that are allowed to have a text data, which keeps a value. Therefore, complex classes have the relationship HASA or HAS with instances of primitive classes to hold



<String namespace=' ' id=' '/>
ISA
<Object namespace='NCBI_gi' id='163483 '/>
ISA
<Integer namespace=' ' id=' '/>
ISA
<VirtualSequence namespace='NCBI_gi' id='163483'>
<Integer namespace='' id='' articleName='Length'>
975
</Integer>
</VirtualSequence>
HASA
HASA
ISA
<NucleotideSequence namespace='NCBI_gi' id='163483'>
<Integer namespace='' id='' articleName='Length'>
975
</Integer>
<String namespace='' id='' articleName='Sequence'>
ATGG...
</String>
</NucleotideSequence>

Figure 9: Classes of objects in BioMoby [WGFS03]

values.

Data classes can have cross-references that point to external data sources. These cross-references help with the integration of the data and services because service providers provide enough cross references to make their services available for more applications. Cross-references also help users find more relevant services to work on their data.

Service ontology has the same hierarchical tree structure as data class ontology. Each service performs a transformation on a data range and produces an output. To introduce a new service in BioMoby, the service should select the type of its service from the service ontology or should introduce a new service type. In addition, the service should specify its input and output data type by choosing from the class ontology.

Finally, the service registry employs a class, service and namespace ontology to search for the appropriate service. Users can search the registry by type of service, the type of input data that the service accepts, or the service signature [WGFS03].

2.3.2.1 A Comparison between BioMoby and Mobyle

Although Mobyle and BioMoby are two different systems, they perform some of the same tasks, yet they use different methods. In the following section, the advantages and disadvantages of these methods are explained. It should be noted that in the following section, MOBY-S as part of BioMoby is compared to Mobyle, since S-MOBY follows a different regime. The following sections will compare the conversion of data types and the quality of services in the Mobyle and MOBY-S.

Conversion of Data Types

Data type conversions have different method in BioMoby verses Mobyle. Conversion of data types in BioMoby is performed by a service that is called upon to work with other services to do the conversions. On the other hand, in the Mobyle, data type conversion is a part of the system and can be performed automatically. However, the squizz-convertor tool that carries out the conversion in Mobyle can be used manually. This eases the user's task, as the user does not need to worry about doing a data type conversion and finding the correct service to accomplish the proper adaptation. In addition, in BioMoby, if a service accepts more than one data type for its input parameters, the service should be registered one time for each of the input types, which is a kind of service duplication [WGFS03]. However, this type of tool or service can be deployed once in the Mobyle environment.

BioMoby uses semantic web technology to describe the data types that are used for connecting services. The ontology terms are kept in the class ontology and each service uses these ontology terms to introduce its input and output data types.

On the other hand, Mobyle asks tools to select their input and output data types among the four groups of data types provided. If a new data type needs to be added to the system, the tool developer should develop a new class that will implement the "DataType" class, directly or indirectly [NMM+09].

Services

BioMoby and Mobyle offer their services in different manners. In BioMoby, a new service can be easily registered. These services may be scattered over different servers and may be executed remotely. It is also possible to connect services to each other, thereby making a chain of services that can analyze a set of data.

Compared to BioMoby, a Mobyle server has some local tools, installed by a Mobyle administrator, to serve users. However, it is possible that a Mobyle portal will request to execute another tool that is installed on a different Mobyle server using the Mobyle Network (as explained in Section 2.3.1.1). The number of the tools installed on a portal is up to a portal administrator.

To compare BioMoby and Mobyle with regards to service efficiency and availability, since BioMoby integrates web services scattered over several servers, the work load is divided between the servers. Therefore, efficiency and availability of BioMoby services will depend on the ones for servers. On the other hand, in the case the tools are installed on only one server in the Mobyle portal, the execution of the tools may slow down by increasing the number of requests. However, the tools on Mobyle are always available, as long as the portal is functional.

2.3.3 Taverna

Taverna [MSRO+10] is a workflow system that is used to assemble complicated, data intensive, web service-based processes. It can be useful in a large variety of data analyses in the life sciences domain, such as proteomic, genomics and structural annotation of biological sequences. It aids biologists who are not very familiar with programming languages. These biologists can make complex workflows by using visual facilities provided by Taverna.

In a Taverna workflow, each software element, such as web services or local scripts, is called “activities” [MSRO+10]. The activities are connected to each other with data links. A “processor”, (representing software components such as web services) is made of several activities. A processor may accept some input from other processers and it can also produce some output for further use in the workflow. A processor can be a complete workflow that carries out a specific function. This ability helps a user to make more complicated workflows that contain other sub-workflows. The sub-workflows can dynamically find a service and can be bound to them at the time of execution. Also, these sub-workflows can be used in a loop and can be performed several times to fulfill an analysis.

In Taverna 2, some fundamental architecture has been redesigned, as explained below [MSRO+10]:

- The old central controller has been replaced with a new system, which runs the processors in parallel with a multi-thread object paradigm. There can be a separate thread to execute each processor, if all of the essential input for a processor is ready.
- When executing each processor, a series of checks are performed to ensure that the quality of the processors’ execution is good. The default dispatch stack is shown in Figure 10 (a). Below, lists of definitions pertaining to checks are provided:
  - “Parallelise” means to evaluate the possibility for a parallel execution of the activities.
  - “Error Bounce” means to check the potential for invalid input data.
  - “Failover” means to discover an event failure of a processor activity and to find another alternative to execute the processor.
  - “Retry” means to retry the execution of a failed activity in a case where the problem was not a fatal error.

– "Invoke", the final step, means to invoke the activity that need to be executed.

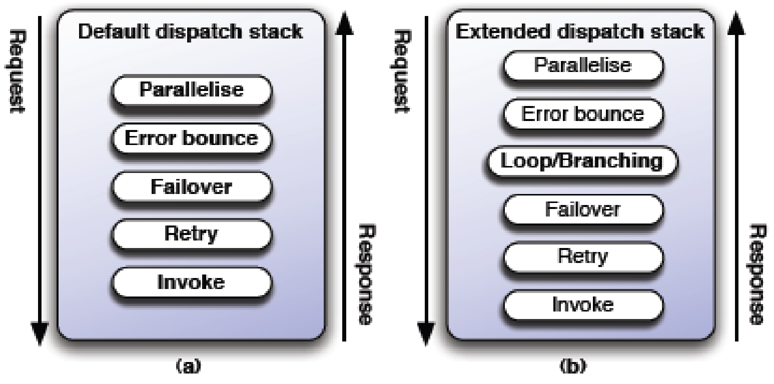


Figure 10: Processor dispatch stack [MSRO$^+$10]

- The above-mentioned dispatch processes could entirely be set up in a Taverna workbench, in the new version of Taverna.

  Finally, the activity will be executed after successfully passing all of the above-mentioned checks.

- Processors may be called several times, using a 'while' loop. As can be seen in Figure 10 (b), another check point is added to the dispatch stack to manage this loop before executing a processor. A loop can be made just on a simple activity or on a complete workflow.

- Taverna supports a plug-in architecture to allow a service provider, such as BioMoby, to implement a special service discovery method and client code. These plug-ins allow Taverna to accommodate the services of those providers.

- Taverna provides the parallelism of inter-processors whenever processors do not have any data dependency with each other. This ability is assured, as a processor cannot be activated when its inputs are not yet available. A new thread can execute a processor that has all of its input ready.

- Intra-processes parallelism is also supported in Taverna. When a list of parameters arrives at the input of a processor that needs a single value, a new processor thread is assigned for each of the values in the list. Each processor thread performs a computation on one of the input values. As the operation on each value may take a different amount of time, the final output is produced only when all of the answers are ready.

- Taverna also supports pipelining of data between several processors. A pipeline of processors can either be "parallel", which means that they do not need to wait for the full output list from the upstream processors, or "serial" which means that they need a fully-computed output list from the upstream processor.

- Taverna keeps data in its memory for whenever a processor needs it, otherwise all of the data is kept on the data server. In the case of services that require intensive data processing, which is popular in bioinformatics, the above-mentioned method results in occupying a small memory space. However, in the instance where a pipeline is made, it is better to always keep data in the main memory, as the output of a processor will be used immediately by other processors. To facilitate this situation, some options have been added to the workbench to control and manage data under different conditions.

- The control loops, such as "while" and "if-then-else", can be made by adding a loop layer to the control steps of the processors.

2.3.3.1 A Comparison between Taverna and Mobyle

Taverna and Mobyle are two systems that can be used to integrate bioinformatics tools and services. Mobyle attempts to simplify user activities by providing as many tools as possible and by connecting these tools easily. On the other hand, Taverna is a system for integrating a large amount of services that are available on the Internet.

Taverna can chain these services to help researchers to accomplish a research study.

In the following sections, the service execution abilities and user facilities of Taverna are compared to those in Mobyle.

Service Execution

Regarding service execution, the first comparison of Taverna verses Mobyle is regarding data pipeline support between the tools and services. Both Taverna and Mobyle support data pipeline. However, in Mobyle, programs can be executed one at a time and the results can be pushed to another program when all results are ready. Services in Taverna, on the other hand, can be executed in parallel if they do not have any dependency to one another [MSRO+10]. Also, it is possible for downstream services in a pipeline in Taverna to start their analyses, even with only a part of data. As a result, computation can be faster in Taverna. Furthermore, if a service needs all output data from the upper stream service, it can wait until the entire amount of required data is ready.

In addition, in Taverna, workflows can be put into a loop, to be executed several times, but in Mobyle, no control loop is provided.

User Facilities

Regarding user facilities, the first comparison of Taverna verses Mobyle is about user work space. The Mobyle portal helps users track their job histories by keeping a workspace for each user. Then, users can have access to the previous jobs and the results so they can be used again. In addition, users can move between the results in order to compare and analyze them [NMM+09]. Taverna does not keep a history of the workflows executed. Instead, a designed workflow can be saved and executed, even outside the Taverna workbench, by using a command line execution tool. In this way, a workflow can be used for more than one study [MSRO+10].

In the context of finding a program to work on the data, Mobyle provides an organized program catalog which can be searched or navigated to find the appropriate program. Moreover, when a tool result is ready, Mobyle automatically suggests a list of tools that can perform further analysis on the result.

On the other hand, it is the user's responsibility to find the required service in Taverna to enact the workflow. However, Taverna has a catalog of services that can be expanded in several ways to accommodate more services. Also, users can search the catalog based on public bioinformatics ontology terms [TMMF09].

Taking all of this information into consideration, although Taverna is more powerful than Mobyle in terms of connecting different services, Mobyle is much easier to work with for a non-computer literate user. As a result, the Mobyle portal was selected as a tool integration environment for this thesis.

The next chapter describes the architecture and design of the "DasSeqFetcher" tool. The DasSeqFetcher is a new tool that was designed to work in Mobyle portal. The tool connects Mobyle to all of the DAS reference servers available on the Internet.

Chapter 3

DasSeqFetcher Software Model

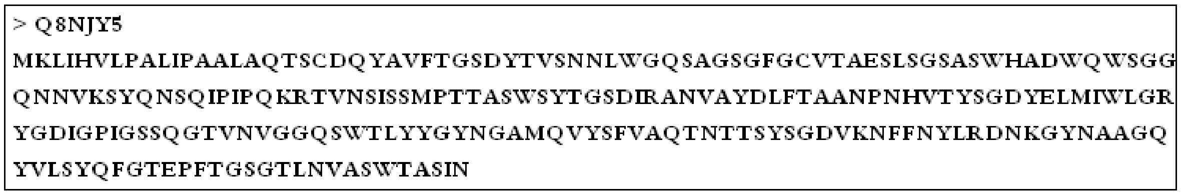
This chapter describes the business rules, requirements specification, architectural design, detailed design and implementation of the DasSeqFetcher tool. The DasSeqFetcher is designed to connect the Mobyle portal to all available DAS reference servers. The DasSeqFetcher is used for retrieving the relevant data from the biological databases via the DAS protocol. The design is presented in UML models [AN05].

3.1 Business Rules for DasSeqFetcher

The DasSeqFetcher is designed to replace the existing manual methods for finding biological sequences from different DAS reference servers. The input data for the DasSeqFetcher tool includes a list of sequence IDs such that the user is interested in their sequence data, along with a list of DAS reference servers that provide the requested sequences.

The user shall enter one or more DNA or protein ID(s), such as “Q8NJY5”, and then shall select one or more DAS reference server names, such as “UniProt”, to fetch the sequence(s). In order to retrieve diverse data from distributed data sources simultaneously, all responses should be merged and returned to the users as one output. Integrating the results simplifies the piping of data to a tool such as the “Multiple Sequence Alignment”, which needs several sequences as an input. If the same sequence is provided by different servers, then only the first encountered sequence should be returned to the user. This way the tool stops searching other servers.

The output is either the retrieved sequence(s) in FASTA format, as shown in Figure 11, or an error message if the parameters are incorrect or a sequence cannot be found on a server.



```
> Q8NJY5
MKLIHVLPALIPAALAQTSCDQYAVFTGSDYTVSNNLWGQSAGSGFGCVTAESLSGSASWHADWQWSGG
QNNVKSYQNSQIPIPQKRTVNSISSMPTTASWSYTGSDIRANVAYDLFTAANPNHVTYSGDYELMIWLGR
YGDIGPIGSSQGTVNVGGQSWTLYYGYNGAMQVYSFVAQTNTTSYSGDVKNFFNYLRDNKGYNAAGQ
YVLSYQFGTEPFTGSGTLNVASWTASIN
```

Figure 11: Protein sequence with sequence ID “Q8NJY5” in FASTA format

3.2 Requirements Specification

This section covers both the functional and non-functional requirements specification for the DasSeqFetcher tool. The former is defined as services that the tool should perform, whereas the non-functional requirements are system constraints provided to mainly meet the integration needs [AN05]. Tables 1 and 2 list the functional and non-functional requirements of the tool, respectively.

3.3 Design

This section describes the system’s architectural design, followed by a detailed design of the DasSeqFetcher tool. An overview of the major subsystems in the DasSeqFetcher is provided. Then, the functionality of the system is described through activity and

ID	**Description**
FR1	DasSeqFetcher shall provide a multiple selection list of DAS reference servers
FR2	DasSeqFetcher shall only show the available reference server(s) from the list of servers
FR3	DasSeqFetcher shall update its list of servers automatically
FR4	DasSeqFetcher shall only use its resources to update the server's list
FR5	DasSeqFetcher shall simultaneously connect to one or more reference servers on each execution
FR6	DasSeqFetcher shall accept one or more sequence IDs to fetch from reference servers on each execution
FR7	DasSeqFetcher shall concurrently retrieve the sequence IDs entered by a user
FR8	DasSeqFetcher shall return the most recent data for the entered sequence ID(s)
FR9	DasSeqFetcher shall check if the user entered at least one sequence ID to be fetched
FR10	DasSeqFetcher shall check if the user selected at least one reference server from the list of servers
FR11	DasSeqFetcher shall search other selected reference severs in case a sequence is not found in the first one
FR12	DasSeqFetcher shall sort the list of reference servers by their nicknames
FR13	DasSeqFetcher shall inform the user about any unavailable sequence(s) by a log message
FR14	DasSeqFetcher shall merge the fetched sequences from several servers into one output

Table 1: DasSeqFetcher functional requirements

ID	**Description**
NFR1	DasSeqFetcher shall use Dasobert APIs
NFR2	DasSeqFetcher shall be a DAS client program
NFR3	DasSeqFetcher shall have an executable format
NFR4	DasSeqFetcher shall be compatible with the Mobyle environment
NFR5	DasSeqFetcher interface shall be described using Mobyle XML interface format
NFR6	DasSeqFetcher shall allow multiple users to access the system simultaneously

Table 2: DasSeqFetcher non-functional requirements

sequence diagrams, which model the system's behavior. More implementation details for the main subsystems are provided at the end of this section.

3.3.1 System Architecture

The DasSeqFetcher is a DAS client application that works in a three-tier architecture and uses the HTTP protocol to communicate with DAS reference servers and the DAS registry. These DAS servers and DAS registry are available on the Internet. The graphical user interface is web-based. The interface has been developed by using the Mobyle portal where the exchange format is the XML files provided by the DasSeqFetcher.

The entire system can be viewed as a layered architecture (Figure 12). The first layer is the *presentation layer* which contains the Mobyle GUI subsystem. As explained in Chapter 2, Section 2.3.1, all tools should provide a Mobyle-based XML file. The XML file provides information to Mobyle regarding how to show the tool's interface as a web page. As a result, considering DasSeqFetcher is a new tool that has been developed to work with Mobyle environment; the DasSeqFetcher should therefore provide an XML file to introduce its interface and parameters to Mobyle.

The second layer is the *business logic layer*. This layer contains the two main subsystems: the ServerList subsystem and the SequenceFetcher subsystem. The ServerList subsystem provides an up-to-date list of all DAS reference servers, sorted by their nicknames. The SequenceFetcher subsystem checks the correctness of the entered data, the availability of selected reference servers and it fetches the sequences from the DAS reference servers. To achieve their tasks, the above-mentioned subsystems coordinate with Dasobert and Mobyle core subsystems. The *Dasobert subsystem* is responsible for sending requests and receiving responses to/from the DAS servers, using the DAS protocol. The *Mobyle core subsystem* is used to integrate the DasSeqFetcher tool with other existing tools in the system (for details, refer to Chapter 2,

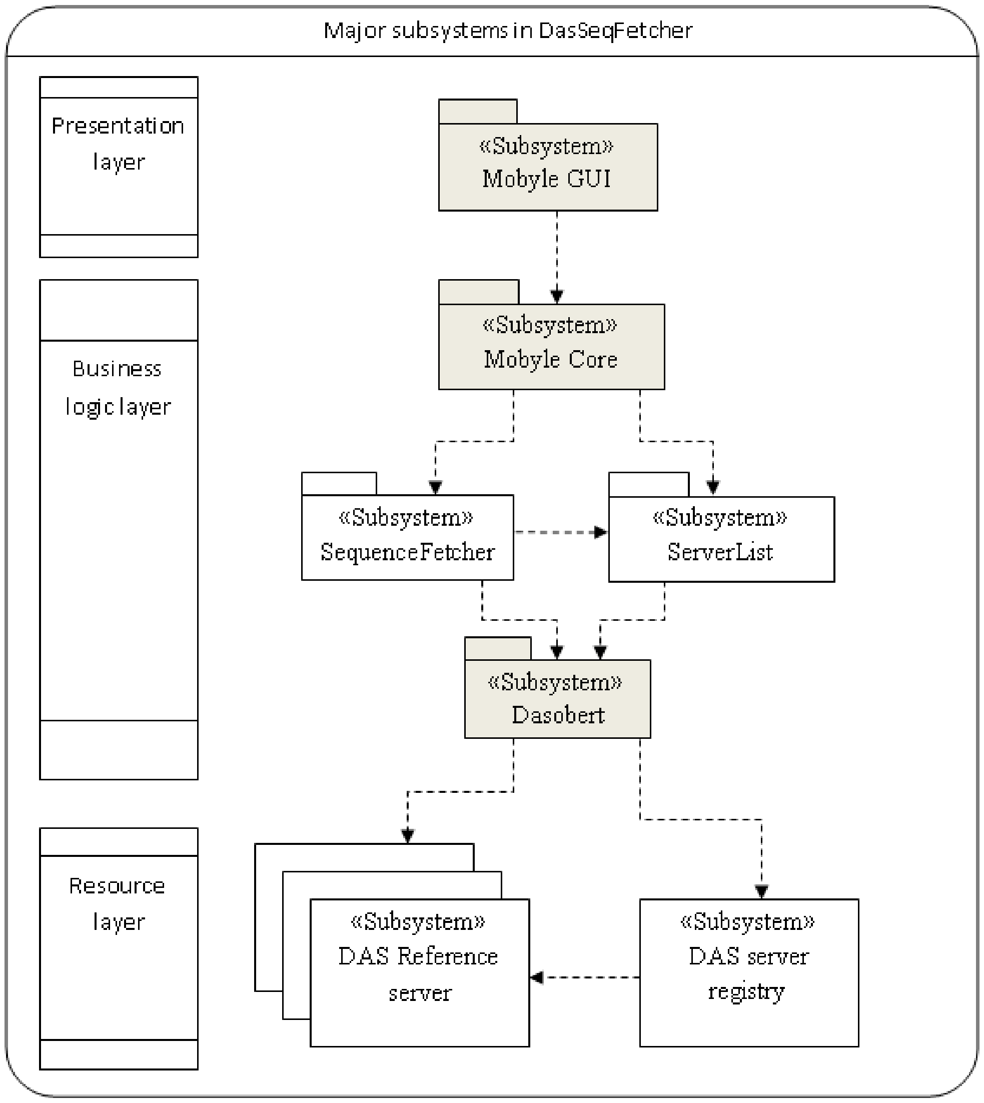


Figure 12: Major subsystems in DasSeqFetcher

Section 2.3.1)

The third layer is the *resource access layer*, where the DAS reference servers that provide the sequence data, and the DAS server registry that provides the DAS servers' data, are both defined.

Figure 13 shows how the subsystems work together. The ServerList is responsible for communicating with the DAS registry. It retrieves the most recent list of reference servers (see Figure 13 , "1"). The ServerList provides a list box from which the users can select servers (Figure 13 , “2” and “3”). On the other hand, the SequenceFetcher subsystem is responsible for getting the requested sequences by connecting to the selected server(s), and by fetching and sending the sequence(s) back to the user (see Figure 13, “5” and “6”).

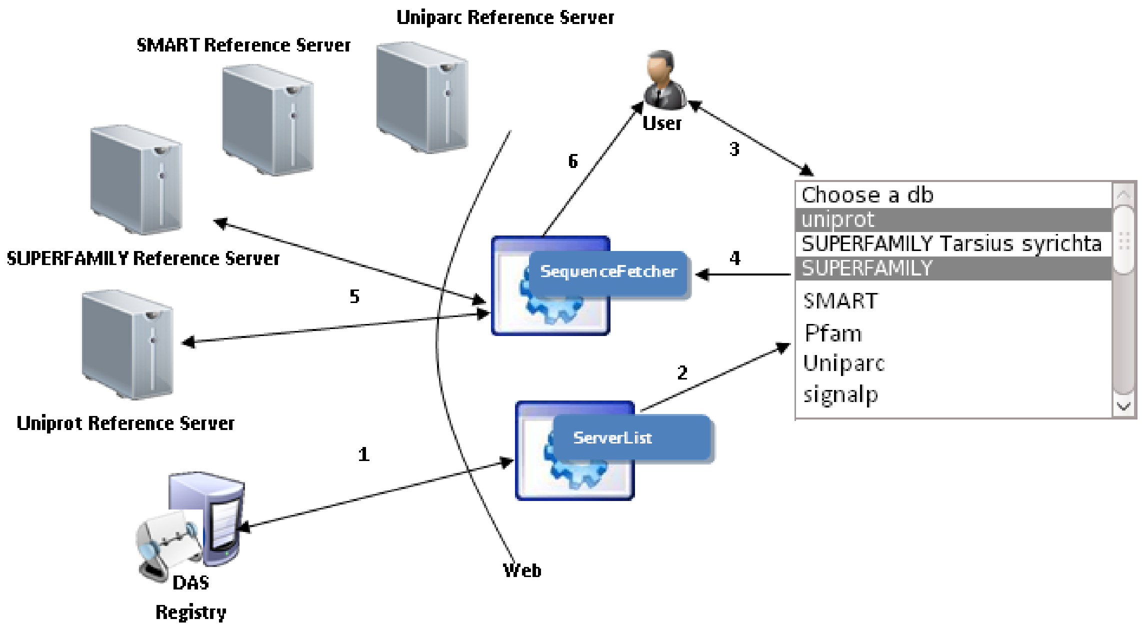


Figure 13: DasSeqFetcher architecture

As subsystems are distributed among different machines, a deployment diagram is used to show another architectural view of the system. Figure 14 demonstrates the assignment of each software artifact to a computational node and the communication method between these artifacts.

Both the ServerList subsystem and the Mobyle portal are web applications that work on an Apache Tomcat web server. The “DasSeqFetcher.Xml” file, which describes the interface of the DasSeqFetcher tool, is also stored in the web server, in order to be accessible by the Mobyle portal. The SequenceFetcher subsystem is a Java application (see Figure 14, “SequenceFetcher.jar”) that is stored in the main server. The clients can use a web browser to connect to the Mobyle portal.

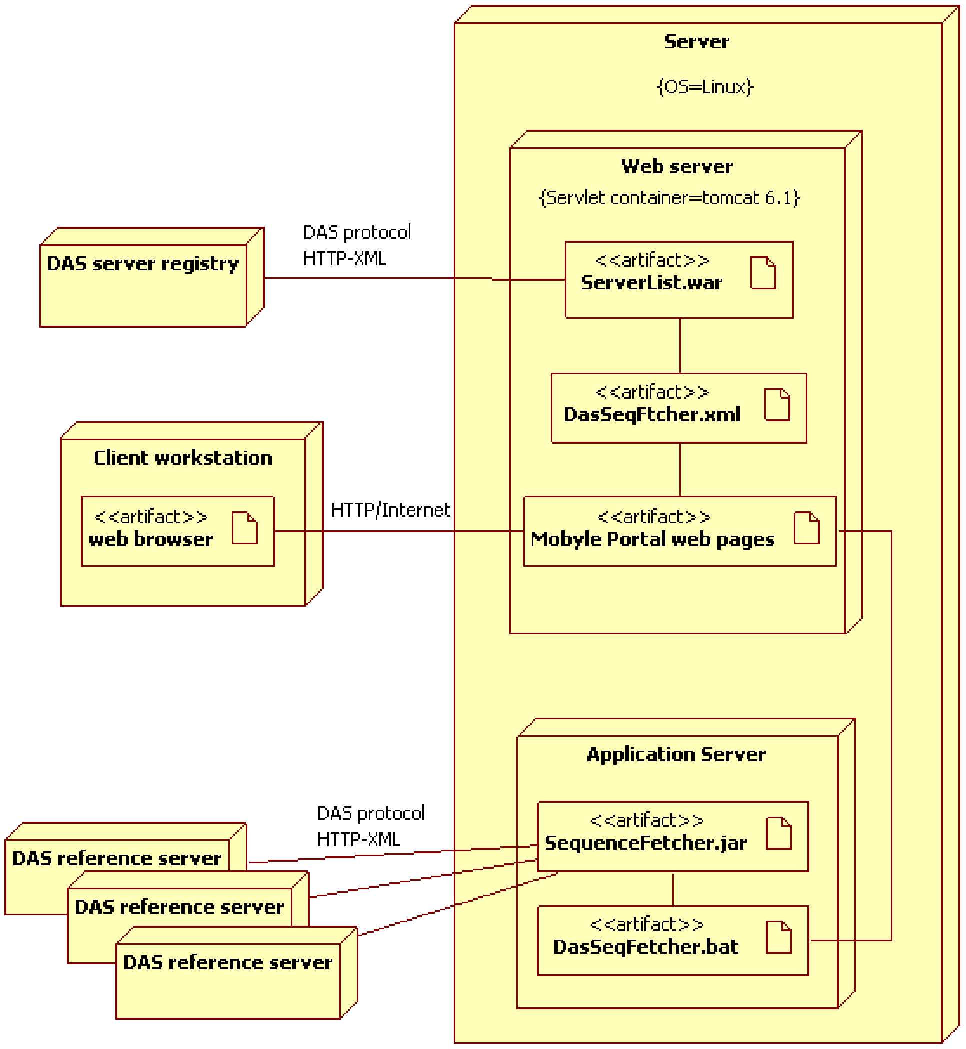


Figure 14: DasSeqFetcher deployment diagram

Mobyle reads the “DasSeqFetcher.xml” file in order to show DasSeqFetcher interface. The list of the servers in “DasSeqFetcher.xml” file is dynamically created by the ServerList subsystem. To run the DasSeqFetcher tool, Mobyle calls “DasSeqFetcher.bat”, which in turn calls “SequenceFetcher.jar” file. Then SequenceFetcher subsystem communicates with multiple servers that are related to different bioinformatics data sources, such as UniProt and Prosite servers.

3.3.2 Detailed Design and Implementation of System

In this section, first we present the graphical user interface design of DasSeqFetcher. The detailed design is then shown, using an activity and sequence diagrams. Two subsystems are presented: SequenceFetcher and ServerList. Finally, this section describes the design and implementation of the SequenceFetcher subsystem first, then it describes the design and implementation of the ServerList subsystem.

3.3.2.1 DasSeqFetcher Graphical User Interface Design

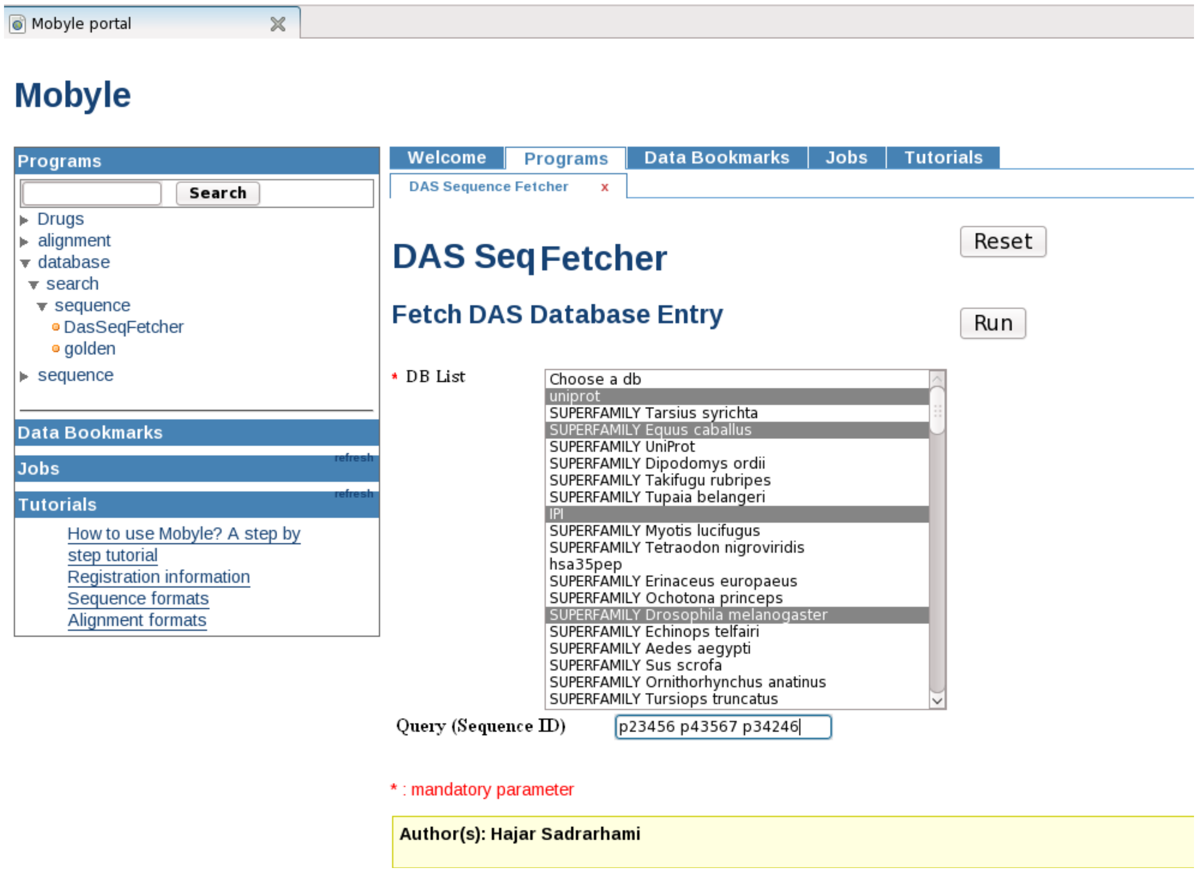
This section illustrates the graphical user interface design of the DasSeqFetcher tool as part of the Mobyle portal. This tool is accessible from the "tools" navigator that is located on the left panel of the Mobyle portal. In Figure 15, the DasSeqFetcher tool is listed under the "sequence" group of search tools intended for databases.

The DasSeqFetcher tool obtains two parameters from the user: the server name(s) and the sequence ID(s). The users can select the DAS servers from the multi-choice list box, named "DB list", and can then enter the sequences ID(s) in the "Query" text box. The sequence IDs in the query text box should be separated by a white space. The users can launch the tool by clicking on the "Run" button at the top. Consequently, the tool shows the retrieved sequence list for the requested sequence ID(s) in the FASTA format in the output window, as shown in Figure 16.

3.3.2.2 SequenceFetcher Subsystem Design

The responsibility of the SequenceFetcher subsystem is to query the DAS reference server(s) selected by the user. It merges the queries results as sequences in the FASTA format, into a single output. The main flow of activities in the SequenceFetcher subsystem is described in Figure 17.

The ServerList subsystem displays a list of servers to a user. The user selects one or more servers from the list, and then the user enters the list of sequence IDs,



DAS Seq Fetcher

Fetch DAS Database Entry

* DB List

```
Choose a db
uniprot
SUPERFAMILY Tarsius syrichta
SUPERFAMILY Equus caballus
SUPERFAMILY UniProt
SUPERFAMILY Dipodomys ordii
SUPERFAMILY Takifugu rubripes
SUPERFAMILY Tupaia belangeri
IPI
SUPERFAMILY Myotis lucifugus
SUPERFAMILY Tetraodon nigroviridis
hsa35pep
SUPERFAMILY Erinaceus europaeus
SUPERFAMILY Ochotona princeps
SUPERFAMILY Drosophila melanogaster
SUPERFAMILY Echinops telfairi
SUPERFAMILY Aedes aegypti
SUPERFAMILY Sus scrofa
SUPERFAMILY Ornithorhynchus anatinus
SUPERFAMILY Tursiops truncatus
```

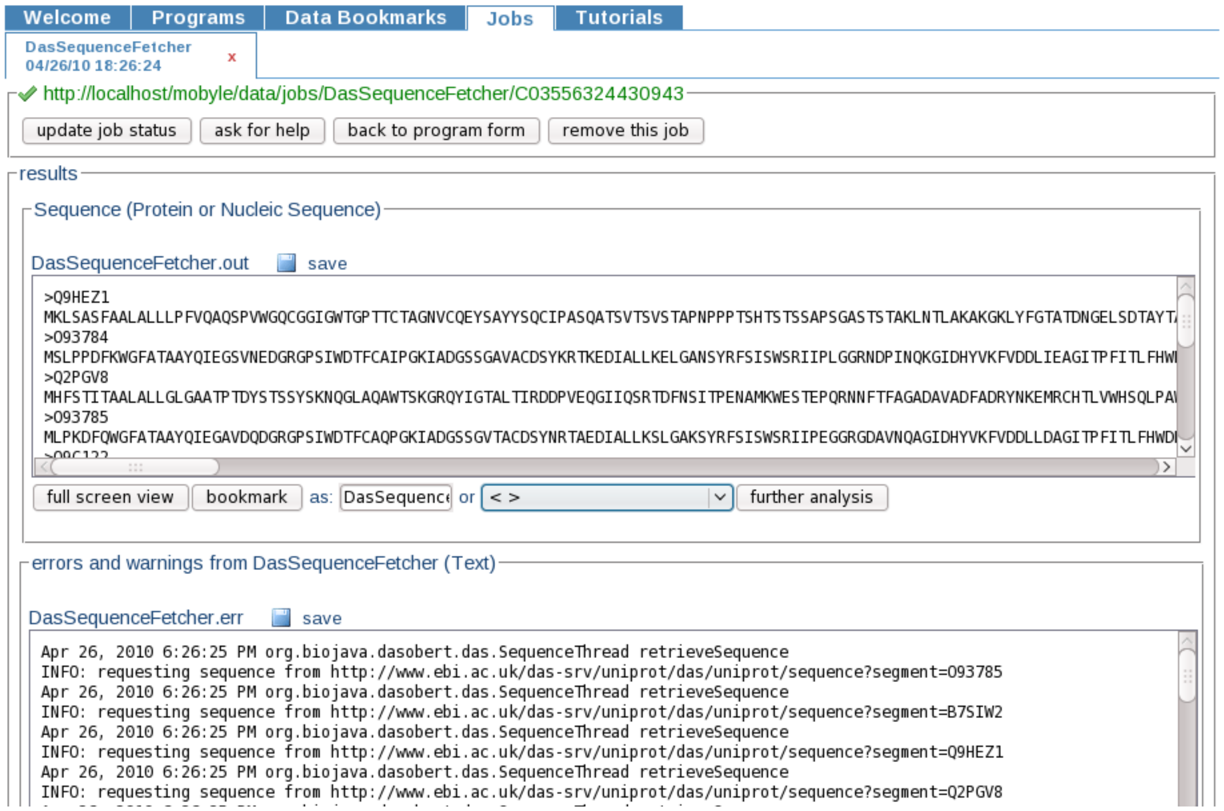
Query (Sequence ID) p23456 p43567 p34246

* : mandatory parameter

Author(s): Hajar Sadrarhami

Figure 15: DasSeqFetcher interface within the Mobyle portal

separated by a space. The SequenceFetcher subsystem creates a new thread for each sequence ID so that the tread fetches the sequence from the selected servers, as shown in the SequenceFetcher sequence diagram (Figure 18). In theory, the number of created threads depends on the number of sequence IDs entered by a user. However, in reality, some other factors can affect this number. As will be discussed in Chapter 4, based on the performed load tests, the system can handle approximately 100 open connections without any problems or errors. Sometimes, when there are more than 100 open connections, the system fails to fetch some of the sequences. For this reason, our confidence load for open connections is 100 connections. In order to cope with this limitation, the SequenceFetcher subsystem creates 100 threads to comply with the number of sequences that can be fetched without any errors. When the 100



```
>Q9HEZ1
MKLSASFAALALLLPFVQAQSPVWGQCGGIGWTGPTTCTAGNVCQEYSAYYSQCIPASQATSVTSVSTAPNPPPTSHTSTSSAPSGASTSTAKLNTLAKAKGKLYFGTATDNGELSDTAYT
>O93784
MSLPPDFKWGFATAAYQIEGSVNEDGRGPSIWDTFCAIPGKIADGSSGAVACDSYKRTKEDIALLKELGANSYRFSISWSRIIPLGGRNDPINQKGIDHYVKFVDDLIEAGITPFITLFHW
>Q2PGV8
MHFSTITAALALLGLGAATPTDYSTSSYSKNQGLAQAWTSKGRQYIGTALTIRDDPVEQGIIQSRTDFNSITPENAMKWESTEPQRNNFTFAGADAVADFADRYNKEMRCHTLVWHSQLPA
>O93785
MLPKDFQWGFATAAYQIEGAVDQDGRGPSIWDTFCAQPGKIADGSSGVTACDSYNRTAEDIALLKSLGAKSYRFSISWSRIIPEGGRGDAVNQAGIDHYVKFVDDLLDAGITPFITLFHWD
```

```
Apr 26, 2010 6:26:25 PM org.biojava.dasobert.das.SequenceThread retrieveSequence
INFO: requesting sequence from http://www.ebi.ac.uk/das-srv/uniprot/das/uniprot/sequence?segment=O93785
Apr 26, 2010 6:26:25 PM org.biojava.dasobert.das.SequenceThread retrieveSequence
INFO: requesting sequence from http://www.ebi.ac.uk/das-srv/uniprot/das/uniprot/sequence?segment=B7SIW2
Apr 26, 2010 6:26:25 PM org.biojava.dasobert.das.SequenceThread retrieveSequence
INFO: requesting sequence from http://www.ebi.ac.uk/das-srv/uniprot/das/uniprot/sequence?segment=Q9HEZ1
Apr 26, 2010 6:26:25 PM org.biojava.dasobert.das.SequenceThread retrieveSequence
INFO: requesting sequence from http://www.ebi.ac.uk/das-srv/uniprot/das/uniprot/sequence?segment=Q2PGV8
```

Figure 16: DasSeqFetcher output

threshold is reached, the subsystem waits for a thread to finish its job and to release the HTTP connection so that the subsystem can create another thread. This process continues until all of the sequences have been fetched. The experiments (see Chapter 4 for details) were conducted on a range of sequences that span from 1 to 10,000. The results confirm the scalability of the DasSeqFetcher.

To fetch a sequence, a thread starts from the first item in the selected server list and the thread queries that server to find the sequence. If it finds the sequence, it produces a "newSequence" event for its listener object, which sends the sequence back to the user and finishes the fetching process. However if the thread cannot find the sequence, then it keep checking the other servers to see if it can find the sequence; otherwise it produces a " notObjectFound" event for the listener object which returns an error message stating that the sequence could not be found.

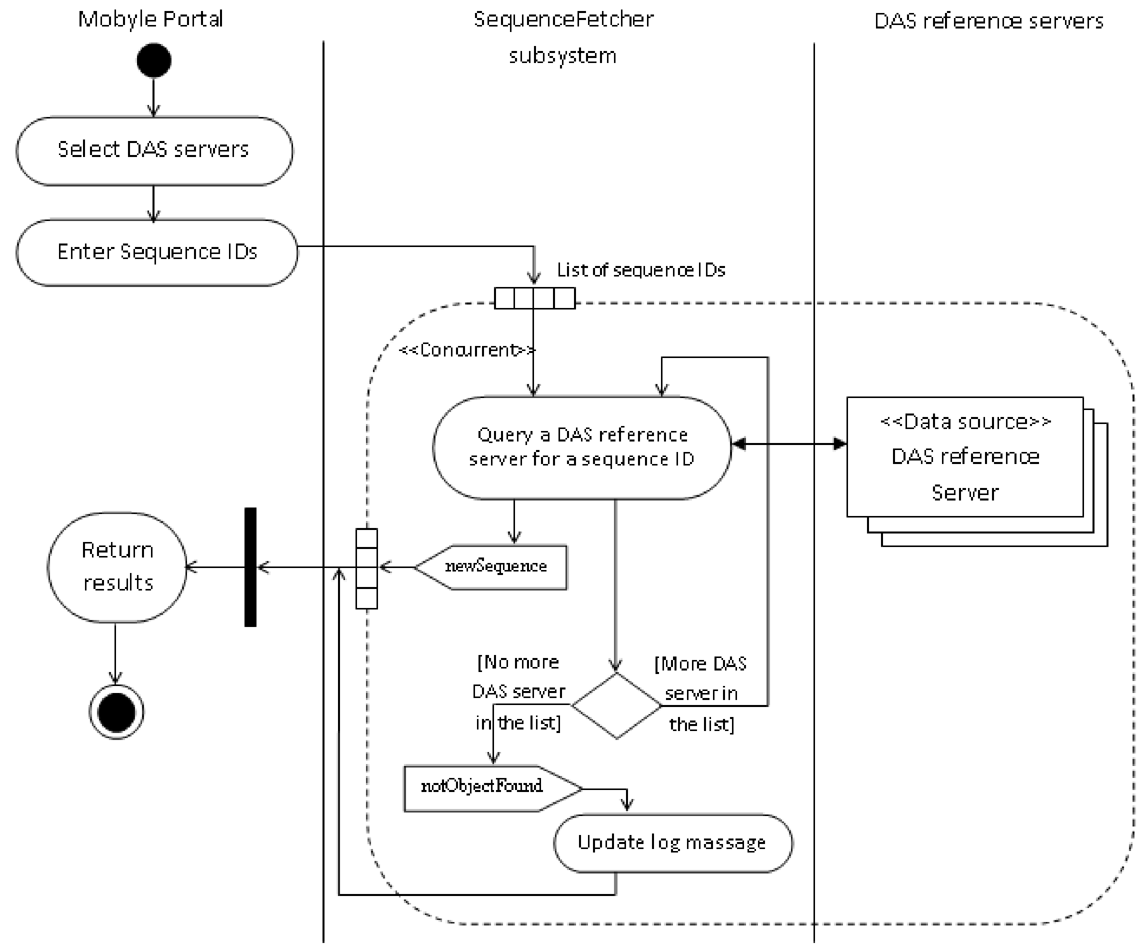

Mobyle Portal
SequenceFetcher subsystem
DAS reference servers
Select DAS servers
Enter Sequence IDs
List of sequence IDs
<<Concurrent>>
Query a DAS reference server for a sequence ID
<<Data source>> DAS reference Server
Return results
newSequence
[No more DAS server in the list]
[More DAS server in the list]
notObjectFound
Update log massage

Figure 17: SequenceFetcher activity diagram

As the threads work concurrently, a lock is implemented to guarantee a mutually exclusive access to the output. This lock prevents threads from displaying partial results on the output. As mentioned in the Chapter 3, Section 3.2, there are two main goals for selecting multiple servers:

1. To have a single output result from all the selected servers

2. To increase the chances of finding a sequence

As mentioned before, the DasSeqFetcher is a multi-threaded program, such that each thread processes its own result whenever the requested sequence is found. This result is immediately displayed for the user on the console. However, if the user is

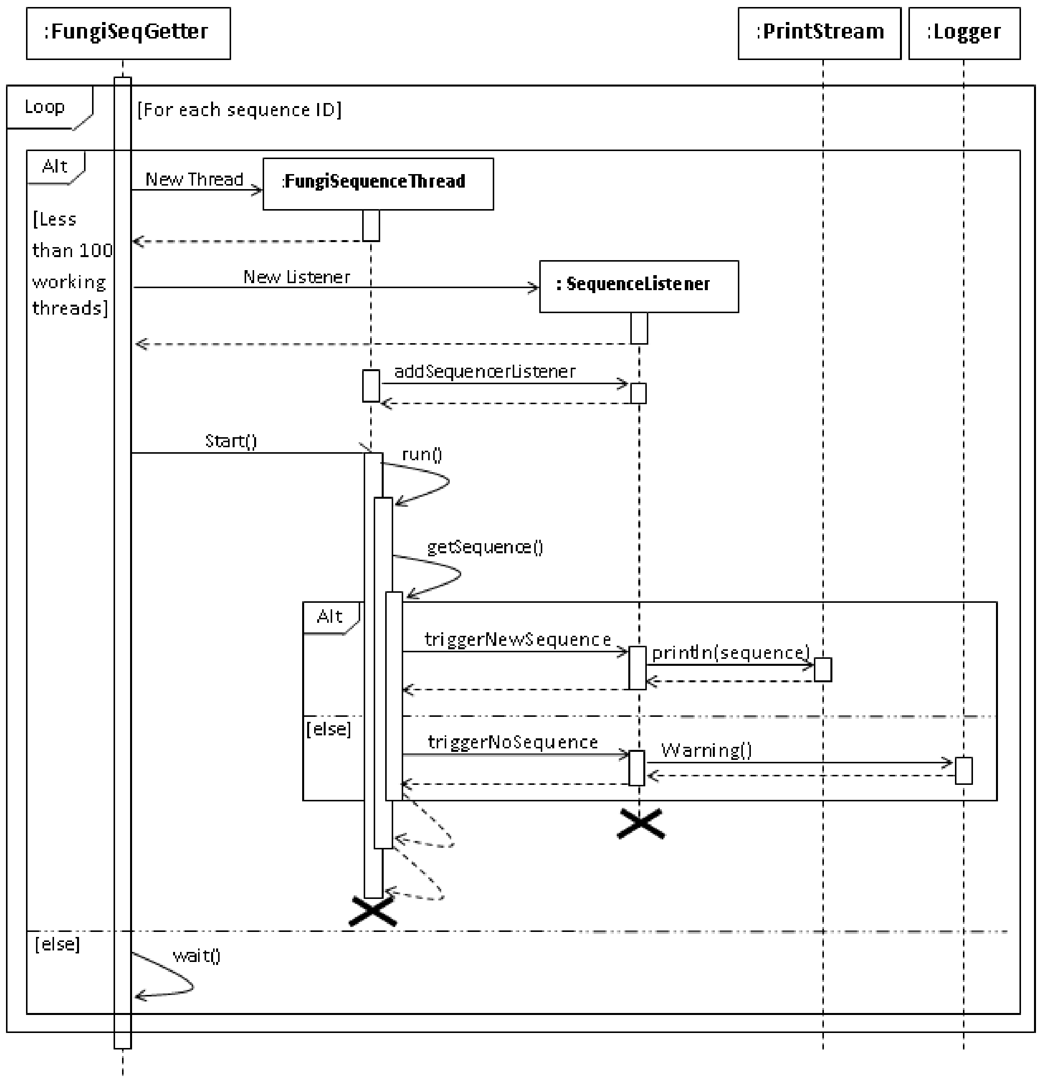


Figure 18: SequenceFetcher subsystem sequence diagram

using a Mobyle user interface, then the final results are displayed once all of the threads finish processing their outputs.

3.3.2.3 Implementation of SequenceFetcher Subsystem

This section explains the implementation details of system threads, using a class diagram. In Figure 19, the object called “FungiSeqGetter” can create as many instances as required of the “FungiSequenceThread” object, each of which has an instance of a listener object waiting for either a “newSequence” event or a “notObjectFound” event.

Each thread starts looking for a sequence on the servers until it finds that sequence. Then, it throws a newSequence event to return that sequence's data. If it cannot find that sequence on any of the servers, then it throws a "notObjectFound" event. The "FungiSeqGetter" object captures both events and displays the appropriate data in the output or log message.

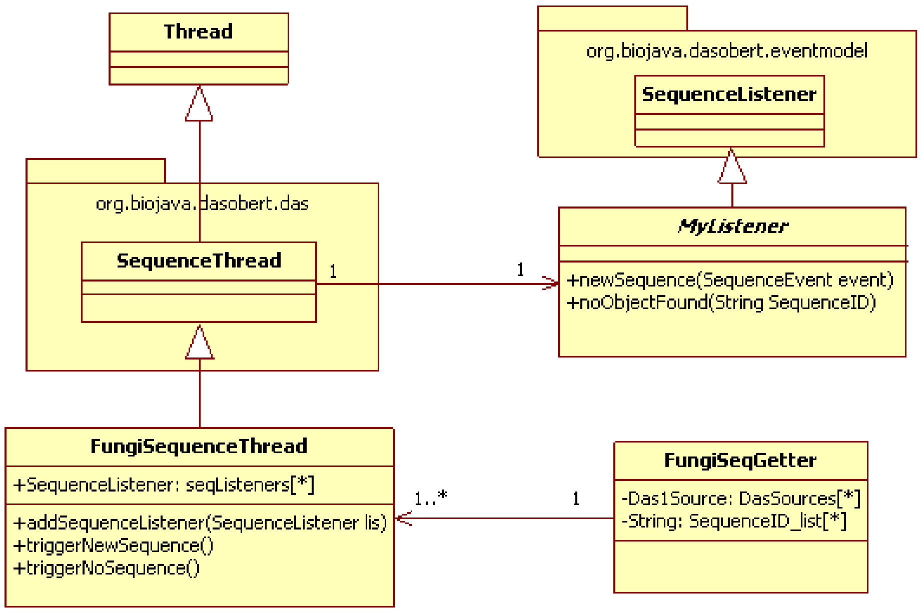

Thread
org.biojava.dasobert.eventmodel
SequenceListener
org.biojava.dasobert.das
SequenceThread
MyListener
1
1
+newSequence(SequenceEvent event)
+noObjectFound(String SequenceID)
FungiSequenceThread
+SequenceListener: seqListeners[*]
+addSequenceListener(SequenceListener lis)
+triggerNewSequence()
+triggerNoSequence()
1..*
1
FungiSeqGetter
-Das1Source: DasSources[*]
-String: SequenceID_list[*]

Figure 19: Class diagram of SequenceFetcher threads

3.3.2.4 ServerList Subsystem Design

The ServerList subsystem is responsible for providing an up-to-date DAS server list. The subsystem connects to the DAS Registry (that can be found at "http://www.dasregistry.org/das1/sources"). Once connected, the subsystem can fetch the recent list of available DAS servers. The subsystem makes an on-the-fly output that has a special syntax. The output has an XML format that follows the Mobyle's program interface convention.

The main activity of the ServerList is called "Load DAS Servers List" (see Figure 20). The DAS registry constantly checks all of the registered DAS reference and annotation servers, which are distributed on different hosts throughout the Internet. The ServerList requests the DAS server registry to return the list of DAS reference and annotation servers, along with their properties, such as their addresses, capabilities, nicknames, registration dates, and coordinate systems. The subsystem then filters out the returned list and selects only those servers that have "sequence" capability, i.e., they are reference servers. The nicknames and addresses of these servers are sorted by the nickname of the servers and then are written in the response.

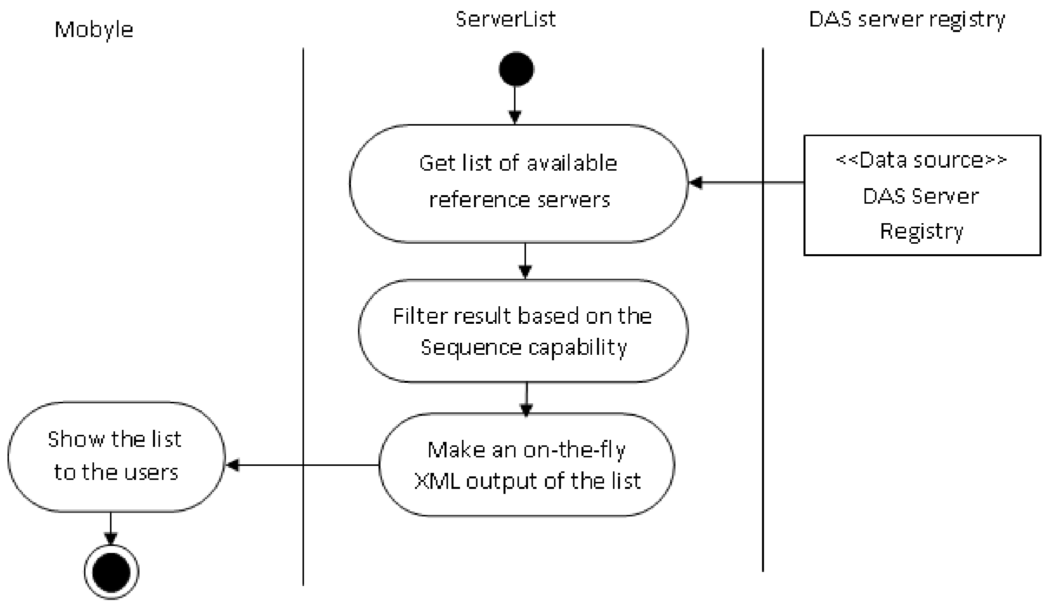

Mobyle
ServerList
DAS server registry
Get list of available reference servers
<<Data source>> DAS Server Registry
Filter result based on the Sequence capability
Show the list to the users
Make an on-the-fly XML output of the list

Figure 20: "Load DAS Servers List" activity diagram

The server list is automatically updated each time that a user loads or refreshes the DasSeqFetcher tool's web page. Therefore, the Mobyle portal administrator does not need to check for any new available updates. In other words, whether a DAS server is down or if a new server is added to the registry, in both cases the server list is automatically updated with the registry data each time that a user refreshes the tool web page.

3.3.2.5 Implementation of ServerList Subsystem

The ServerList subsystem has been implemented as a Java servlet. This subsystem uses the Dasobert API [Prl07]. Figure 21 shows a part of code that has been taken from "DasSeqFetcher.xml" file (The DasSeqFetcher XML interface). The file is responsible for sending the request to the ServerList subsystem and includes the results. A full description of the "DasSeqFetcher.xml" file can be found in Appendix A.

```
<xi:include href="http://compute.fungalgenomics.ca /DasServerList/ServerList">
 <xi:fallback>
    <vlist>
      <velem undef="1">
         <value>null</value>
            <label>Choose a database</label>
         </velem>
     </vlist>
  </xi:fallback>
</xi:include>
```

Figure 21: Part of the DasSeqFetcher XML interface file

When using an "include statement", Figure 22, a browser makes an HTTP request for the address to the corresponding web server.

```
<xi:include href="http://compute.fungalgenomics.ca /DasServerList/XmlServerList">
```

Figure 22: Include statement used to call the XmlSeverList servlet

With the help of a servlet, the URL pattern (/XmlServerList) is mapped to a servlet class. This class will be invoked whenever a request is made for the above URL. Figure 23 shows a sample output from the ServerList subsystem in the Mobyle XML format.

Figure 24 shows how the "DasSeqFetcher.xml" file makes a request to the ServerList servlet program (Figure 24, "1") in order to produce the list of servers as a response (Figure 24, "2"). The response is automatically embedded to the "DasSeqFetcher.xml" file (Figure 24, "3") which is then used by Mobyle to create the tool's user interface (Figure 24, "4").

```
<velem>
  <value>http://www.ebi.ac.uk/das-srv/uniprot/das/uniprot/</value>
  <label>UniProt</label>
</velem>
<velem>
   <value>http://proserver.vital.it.ch/das/prositefeature/</value>
   <label>Prosite Features(matches)</label>
<velem>
```

Figure 23: A part of output from the ServerList subsystem

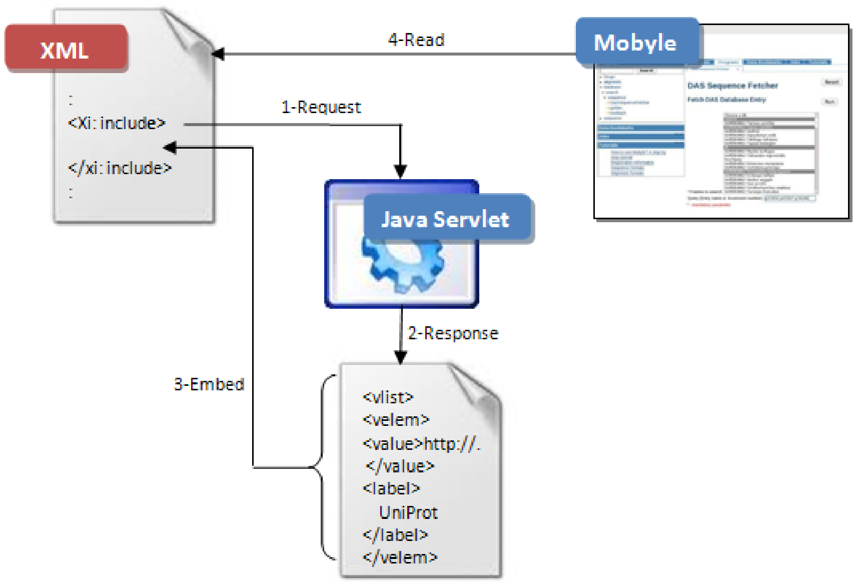


Figure 24: Updating server list in “DasSeqFetcher.xml” file

3.3.2.6 Challenges in Design

This section describes several attempts to create a dynamic list of DAS servers. The above-mentioned version of ServerList subsystem is found to be the best solution in terms of meeting the system’s functional and non-functional requirements.

In order to create the server list, first of all, a Java application is written to connect to the server and fetch the list. Although this program has functioned correctly, it was not a dynamic web application. This program also needed an operating system script to run it on the disk every time that the user needs to update the XML file of the server list. The main XML interface file was then able to read the data from server list file and make the list box appear on the web page.

The problem with this method is that the program does not provide the most up-to-date data. For instance, in a case where the shell script ran the program every 5 minutes, if during this period the server list on the DAS registry was changed, e.g., a crash of a DAS server, then the user would get an invalid server, and therefore a run time error would occur. Also, the program depends on a shell script to update it when needed. As a result, this method does not meet R3 requirement in Table 1.

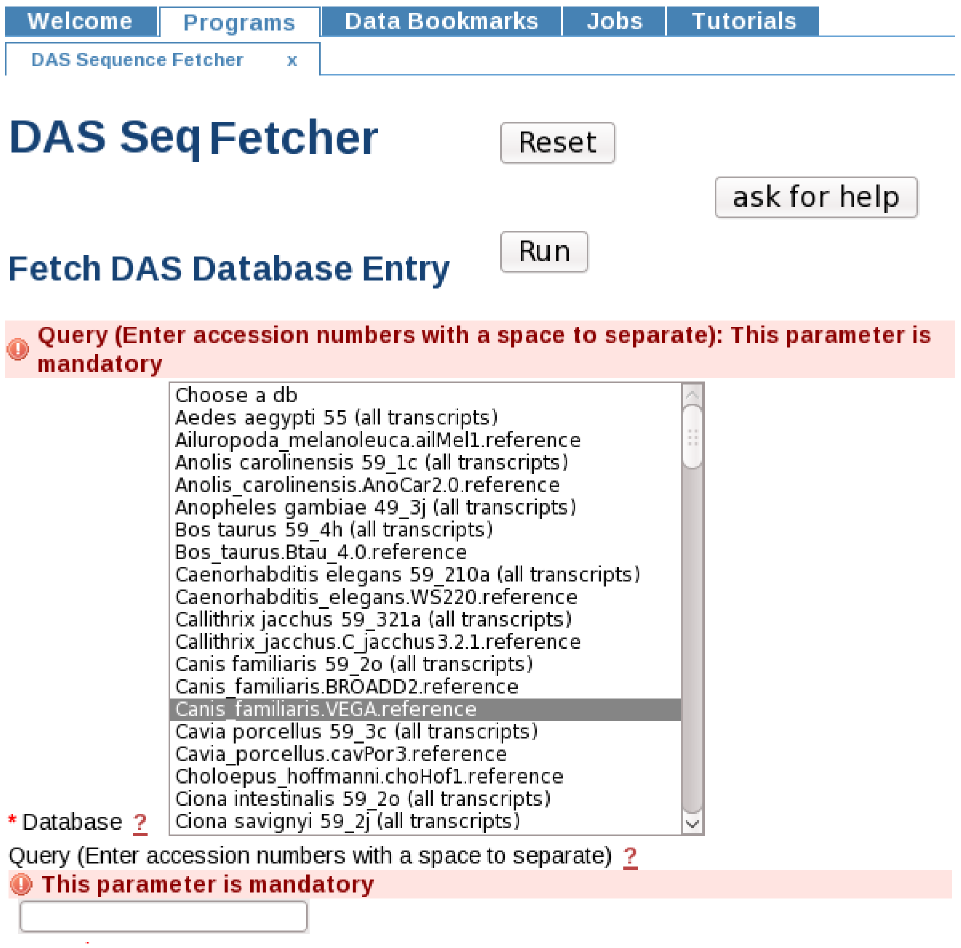
A second attempt was made to write a Java applet so that it could request data from the DAS registry once the user had access to the program's webpage. However, requesting data from the DAS registry in this program is a "server-side" activity, i.e., it does not need to get any information from the user to proceed and the entire job is performed on the server side. As a result, a client-side applet is obviously not a good choice in this situation. Another problem with implementing an applet is that in a case such that the web server works behind a firewall, the applet would need to change the system properties, which is not allowed.

The last strategy for providing an up-to-date server list uses dynamic web programming. It was adopted for the application used in this thesis and addresses all of the previously described problems. This technique returns the most recent server list to the user at all the time. Using this method as a server-side code, there is no security concern pertaining to change the system properties to pass the firewall.

3.3.2.7 DasSeqFetcher Tool Error Detection and Validation

There are two stages for error detection and validation in the DasSeqFetcher tool. The first step in error detection is at the browser level. This level uses the parameter dependency control of the Mobyle, which is the one described by the DasSeqFetcher in the XML interface file. Therefore, the data is validated before being sent to the server for processing. In the case of an error situation, the message is shown in red text next to the Databases list or the Query text box, as shown in Figure 25. The second stage of validation takes place at the server level, when the server processes the data. In case of any error, error messages are shown in the "DasSeqFetcher.err" box on the Mobyle result page, as was shown in Figure 16 of Section 3.3.2.1.

At the time of this thesis the DasSeqFetcher tool has been plugged into the CBN Mobyle server at the Loyola campus of Concordia University where biologists have been using it to connect to available DAS reference servers. The DasSeqFetcher allows biologists to fetch the required protein and DNA sequences.



Query (Enter accession numbers with a space to separate): This parameter is mandatory

* Database ?

Choose a db
Aedes aegypti 55 (all transcripts)
Ailuropoda_melanoleuca.ailMel1.reference
Anolis carolinensis 59_1c (all transcripts)
Anolis_carolinensis.AnoCar2.0.reference
Anopheles gambiae 49_3j (all transcripts)
Bos taurus 59_4h (all transcripts)
Bos_taurus.Btau_4.0.reference
Caenorhabditis elegans 59_210a (all transcripts)
Caenorhabditis_elegans.WS220.reference
Callithrix jacchus 59_321a (all transcripts)
Callithrix_jacchus.C_jacchus3.2.1.reference
Canis familiaris 59_2o (all transcripts)
Canis_familiaris.BROADD2.reference
Canis_familiaris.VEGA.reference
Cavia porcellus 59_3c (all transcripts)
Cavia_porcellus.cavPor3.reference
Choloepus_hoffmanni.choHof1.reference
Ciona intestinalis 59_2o (all transcripts)
Ciona savignyi 59_2j (all transcripts)

Query (Enter accession numbers with a space to separate) ?

This parameter is mandatory

Figure 25: DasSeqFetcher validation error message

More details about the installation instructions of the tool to be used on a Mobyle portal can be found in Appendix B.

Chapter 4

DasSeqFetcher Evaluation

In this chapter, the DasSeqFetcher is evaluated as per the requirements in Chapter 3. This chapter will show how the DasSeqFetcher can fulfill those requirements as a data aggregator tool. A load test has been performed (see Section 4.2) in order to evaluate the tool's performance. In addition, the DasSeqFetcher is compared to similar tools and the pros and cons are discussed (see Section 4.3).

4.1 Evaluation Pertaining to Requirements

Below, the DasSeqFetcher is evaluated pertaining to the requirements mentioned in Chapter 3.

list of server items: To satisfy the R1 and R12 requirements in Table 1 of Chapter 3, the DasSeqFetcher connects to the DAS registry to collect and then show all of the valid DAS reference servers in a multiple selection list box. To simplify, the searching of the list for users, the list box is sorted based on the servers' nicknames.

Adaptability: As mentioned in Chapter 3, the DasSeqFetcher server list is updated automatically, so there is no need to run a separate program when the tool web page is reloaded. For instance, if the UniProt DAS server is broken for awhile or fails to provide the "sequence" capability, then the DAS registry deletes or changes the status of the UniProt DAS server on the registry. If the reference server is deleted from the registry, the registry does not return the UniProt as an operating server. As a result, UniProt will not be displayed in the server list. Moreover, if the UniProt fails to provide sequences, it will be removed from the server list, while the DasSeqFetcher queries the returned list from the DAS registry. And such queries are based on the DAS reference servers' capabilities. This process ensures that the list of servers does not show unavailable servers. These provisions guarantee that the requirements R2, R3 and R4, given in Chapter 3, Table 1, are satisfied.

Merging the searches and results: The DasSeqFetcher removes the burden from the users by searching each DAS server, one by one, in order to find the data that is required to be transferred to Mobyle. As is demonstrated in Figure 26 (a), without using DasSeqFetcher, a user has to spend a tremendous amount of time to obtain all of the sequences required. However, as shown below in Figure 26 (b), the DasSeqFetcher program has the capability to perform all of the steps, including finding the data sources, aggregating the data available on different servers, and loading data as a single output into the Mobyle. This functionality covers the R14 requirement.

Concurrency: As per the R5, R6 and R7 requirements regarding requests for multiple sequence IDs from several servers in a tool execution, the DasSeqFetcher enables users to select more than one server from the server list and to enter a list of sequence IDs to be fetched. As mentioned in Chapter 3, Section 3.3.2.2, the DasSeqFetcher performs this service by employing a thread for each sequence ID that the user enters. The concurrency enabled by the use of threads allows computations to continue in one thread while another may be waiting for a response from a data source. As a result, the program achieves the maximum concurrency.

Timeliness: In terms of the requirement for returning the most up-to-date data (R8, timeliness), the DasSeqFetcher users can fetch the most recent sequences, since

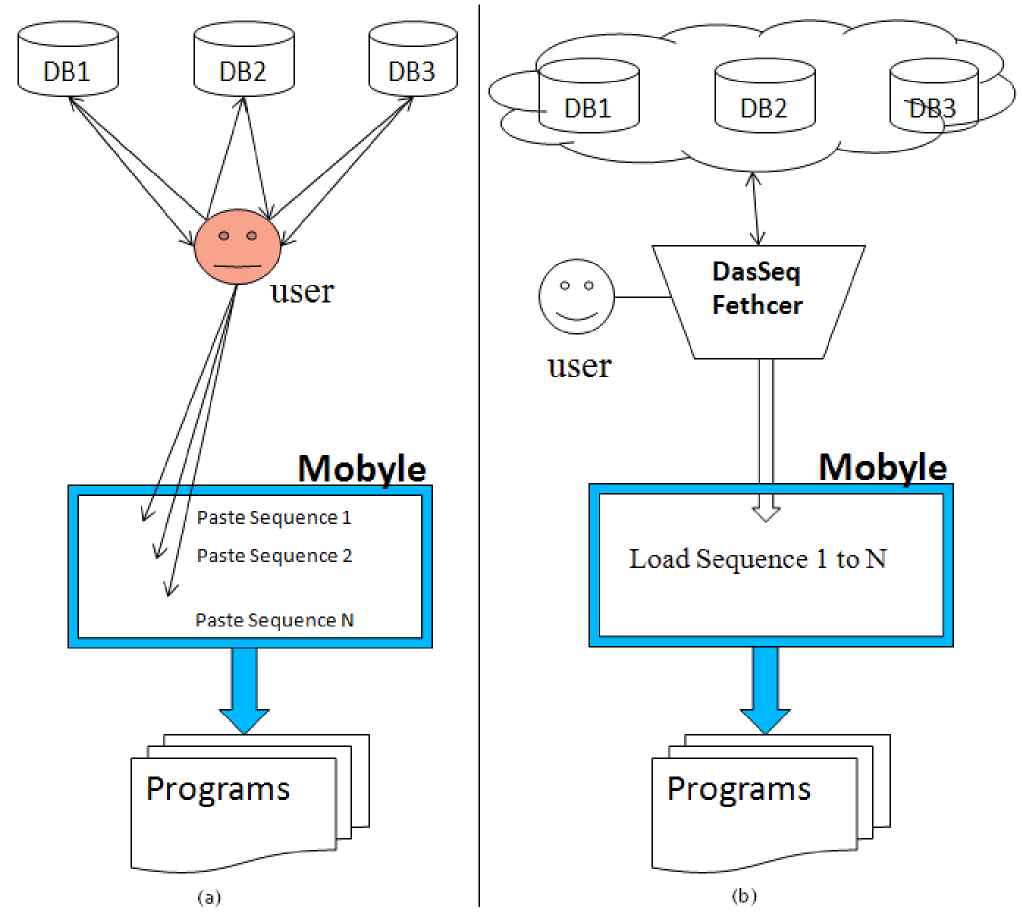


Figure 26: Comparison of users' workloads without (a) and with (b) using the DasSeqFetcher

they can connect directly to the DAS servers that provide the sequences. For instance, if new information on a protein sequence is added to a database, and if a user happens to request that sequence right after it is placed in the database then the DAS reference server can return that sequence pertaining to the requested sequence ID. This approach contrasts other approaches such as the "data warehouse" approach, that needs awhile to aggregate the new data from the main data sources.

Robustness: The R9, R10, R11, and R13 requirements in Table 1, regarding the robustness of the DasSeqFetcher, can all be evaluated from three different perspectives.

First of all, if at the exact moment of querying a server it becomes unavailable, then the system will return a warning message to the user. It shows that the system is robust enough to prevent possible error in this situation.

Another issue of robustness that is addressed by DasSeqFetcher is the interface robustness. The DasSeqFetcher checks whether a user has entered at least one sequence ID and has selected one DAS reference server from the list, to prevent any errors. If a user does not provide enough data, then the program warns the user about the problem. In addition, the tool checks whether or not the data entered follows the correct format.

Finally, if a DAS reference server does not contain the requested sequence, or if the reference server does not reply to the request, then the DasSeqFetcher searches through other reference servers from the user server list selection to eventually find the sequence. For example, a user can search for sequence ID "P23456" provided by the UniProt DAS server. However, if the user also selects the IPI DAS reference server, which does not contain this sequence ID, the thread starts sending the request to the IPI server and waits for the reply. Since this server does not have the sequence, then it replies with a warning message stating that it "did not find any sequence with the sequence ID P23456". The thread will then send another request to the UniProt DAS reference server that does contain the sequence. Finally, the UniProt returns the sequence for the "P23456" ID.

Non-functional requirements: To satisfy non-functional requirements, the DasSeqFetcher, written in Java, uses the Dasobert API to query the DAS registry and reference servers. Also, the DasSeqFetcher has an executable version and an XML interface file that makes the tool compatible with the Mobyle portal. Moreover, as the DasSeqFetcher is a web application, multiple users can use it simultaneously by accessing it in the Mobyle portal.

In the next section the system's behavior will be analyzed by using a load test.

4.2 Performance Evaluation

To verify the performance of the DasSeqFetcher, a load test was conducted by using different amounts of sequence IDs and by using at least two well-known DAS servers. The response time of this test by using DasSeqFetcher, was then compared twice: first, it was compared with the average time consumed, when the same test was performed manually, as well as when using a Linux shell script. The results are summarized below in Table 3.

It should be noted that the maximum number of sequences in datasets used by the CBN group was about 10,000. The system was then tested using 1, 10, 100, 1000, 1500 and 10,000 sequences. The test set used to be tested by the system contained protein sequences from the UniProt and Prosite DAS reference servers. The operating system used to perform the test was Linux, installed on an "Intel@Core2" processor with 2 MB Cash and a CPU speed 1.80GHz. For each set of sequences, the response time was recorded on three different trials. Also, an average processing time of 30 second per sequence was estimated when the process was performed manually by a user. This manual process was performed by making a query for each sequence ID, followed by sending the query to a DAS server using a web browser. After receiving the sequence result, the user was able to store the result in a file to be used for the subsequent processes; for instance, to load the result into the Mobyle portal. we were assumed that the address of the DAS reference server had already been found.

Also, a Linux shell script was written in order to test the performance of a stand-alone program that could fetch the sequences from the UniProt DAS reference server. The script reads the sequence IDs from a text file and queries them, one after another, from the UniProt DAS reference server using the "wget" command. The shell script is shown in Figure 27, below. The result of this shell script is shown in Table 3.

```
while read FILE; do
wget "http://www.ebi.ac.uk/das-srv/uniprot/das/uniprot/sequence?segment="$FILE"";
done < ../sequenceIDs
```

Figure 27: The shell script used to fetch the test dataset sequences from the UniProt DAS server

	Response time (milisecond)								
	Using DasSeqFetcher							Linux Script	Manually estimated
	As many threads as required						100 threads		
Server Name	UniProt			Prosite			UniProt	UniProt	UniProt
No of seq	Trial 1	Trial 2	Trial 3	Trial 1	Trial 2	Trial 3			
1	511	447	530	653	547	583	512	1×10^3	3×10^4
10	734	594	505	736	613	664	549	3×10^3	3×10^5
100	940	841	995	1703	1612	3402	856	23×10^3	3×10^6
1000	5857	11498	4158	PR*	PR	PR	7480	235×10^3	3×10^7
1500	PR	14085	5884	PR	PR	PR	12358	349×10^3	4.5×10^7
10,000	-	-	-	-	-	-	65634	24×10^5	3×10^8

Table 3: DasSeqFetcher, Linux shell script and manual method response times for different numbers of sequences (PR, Partial Results: the tool did not return the results for some of the sequences)

Based on the results shown in Table 3, we conclude the following:

First, the response times using the DasSeqFetcher were considerably shorter than those obtained with the manual method or the shell script. This result demonstrates that the DasSeqFetcher can clearly save biologists time. Therefore, the DasSeqFetcher ensures that their research is more efficient as they can find the required data in less time.

Second, even for the same number of sequences and servers, response times were variable, depending on the DAS servers' workload. For example, for 1000 sequences, the response time ranged from about 3-15 second for UniProt. It should be noted that the time spent by the DasSeqFetcher to process the returned sequences was negligible when compared to the aforementioned DAS servers' response time.

Moreover, the tested servers could support only a limited number of simultaneous requests for sequences from the same IP address, probably in order to continue supporting a larger amount of users. For UniProt, this situation happened where about 1500 sequences were requested concurrently from the same IP address. The same test was performed on the Prosite DAS reference server. It was observed that the server refused to accept a new request after responding to 800 sequences. These numbers are also not absolute and may vary, based on the performances of UniProt and Prosite DAS reference servers. Therefore, the workload of the server(s) was found to be a key parameter affecting the DasSeqFetcher performance.

Third, as the number of sequence IDs increased to 1024, an error was raised, showing that the default number of open TCP connections in Linux is limited to 1024. To solve this limitation, we decided to close all of the connections to a DAS reference server that were no longer needed.

Fourth, as can be seen in Table 3, there is no linear relation between the number of requested sequences and the response time. In fact, when the number of requested sequences grows quickly, the response time increases only slightly. This shows the advantage of using the DasSeqFetcher, in which the threads work concurrently.

One approach to solving the problem (regarding a limited number of connections to a DAS server) is to ask for more than one sequence in each query, in order to reduce the number of concurrent client connections. The problem is that the DAS 1.5E, which is used by almost all available DAS reference servers, can return the requested sequences only if all are available on the target DAS server. In other words, if one sequence is not available on a DAS server, the server cannot return the other available sequences. It should be noted that this problem has been solved in the new version of the DAS protocol, version 1.6. However, at the moment, only a low number of DAS servers support this version.

Another solution is to send a constant number of requests simultaneously. Based on trial and error experiments, the above number was optimized to be one hundred requests without producing any errors. This approach may require longer time to fetch all of the sequences from the data sources because some of the executions of requests may be postponed. The postponed requests wait for the working ones to finish. However, all of the requested sequences are eventually fetched, even with the number of sequences that exceed the limitations of the DAS reference servers, which is attributed to the DasSeqFetcher's scalability. As was shown in Chapter 3, Table 1, by using this method, the tool can fetch 10,000 sequences in only 65634 milliseconds or in approximately 66 seconds. The results show that the process of fetching 10,000 sequences is faster by the DasSeqFetcher than with the other two previously discussed methods (manual and Linux shell script).

One major lesson learned from this study is that the performance of a new program, when written in a distributed environment, depends not only on the program's quality but also on the service quality of the server with which the program communicates. This situation is mainly a result of the nature of such systems that are offered by different data providers, which is out of the programmer's control.

In the following section, the features of DasSeqFetcher are compared to those of SeqretFromUSA and Golden, as two available data aggregator tools in the Mobyle Portal.

4.3 Comparison with Similar Tools

Mobyle, as discussed in Chapter 2, Section 2.3.1, uses two data aggregator tools, namely SeqretFromUSA and Golden. These tools are used to import the data required by biologists into the Mobyle portal. While acceptable for fetching sequence data, these tools do not meet the requirements that are important for a DAS data fetcher tool.

Similar to DasSeqFetcher, SeqretFromUSA can query more than one server for a list of sequence IDs at the same time. However, it does not provide a list of databases; therefore, a user has to be familiar with the databases in order to enter their names. This is not a user-friendly approach and the R1 requirement in Table 1 of Chapter 3, cannot be fulfilled. Generally, there is no standard way for selecting an EMBOSS (European Molecular Biology Open Software) database name [RLB+00]. A name is selected by the system administrator, which complicates the task for a user even more.

In addition, SeqretFromUSA reads the sequences from local databases, which rely upon the system administrator to update their data, whereas DasSeqFetcher connects directly to the main databases, always fetching the most up-to-date data.

On the other hand, Golden differs from SeqretFromUSA and DasSeqFetcher in that it is able to fetch only one sequence at a time from only one of the well-known bioinformatics databases. The Golden database list contains a limited and constant number of databases that are already installed on the local server, and this list can only be upgraded by the system administrator. As a result, users do not have access to the most up-to-date data. Also, if a tool in the Mobyle portal needs more than one sequence, and the sequences are available on several databases, Golden cannot retrieve multiple sequences and merge the results into one output. Thus, Golden cannot cover the R5, R6 and R7 requirements.

In addition, neither of these tools (SeqretFromUSA and Golden) have access to the DAS servers, resulting in the loss of a major data source. A feature comparison between DasSeqFetcher and the two aggregators is shown in Table 4, below.

	DasSeqFetcher	SeqretFromUSA	Golden
Connecting to DAS reference servers	yes	no	no
Automatic data source list update	yes	no	no
Ability to fetch more than one sequence	yes	yes	no
Connecting to more than one server	yes	yes	no
Dynamic number of data sources	yes	no	no
Connecting to distributed databases	yes	no	no
Providing most up-to-date data	yes	no	no

Table 4: DasSeqFetcher comparison with similar data fetcher tools

Chapter 5

Design of mycoCLAP DAS reference servers

This chapter describes the software design of mycoCLAP DAS reference severs. These servers were designed to provide the DNA and protein sequences that have been stored in the mycoCLAP database [MWP+10]. This database contains information on characterized lignocellulose-active proteins, produced by the CBN (Cellulosic Biofuel Network) group.

As the CBN project is a large-scale research project, accessing information in the mycoCLAP database is a vital issue for all group members. An effective way to deal with this issue, according to the literature [DJD+01, PDK+07], is to employ the DAS protocol, in order to make the fungal protein and DNA sequences available to all parties who are interested.

As a part of this thesis, two DAS reference servers were developed, to provide information on protein and DNA sequences, stored in the mycoCLAP database. This chapter explains the architecture and design of the mycoCLAP DAS reference servers. The high-level architecture of the system is presented, and a deployment diagram is explained, followed by an explanation of the design of both the system and the database.

5.1 Requirements Specification

This section provides details on the requirements for the mycoCLAP DAS reference servers, including both the functional and non-functional requirements.

ID	**Description**
FR1	mycoCLAP DAS DNA/protein reference servers shall accepts a user defined sequence ID as part of server query URL
FR2	mycoCLAP DAS DNA/protein reference servers shall return the requested DNA/protein sequences
FR3	mycoCLAP DAS DNA/protein reference servers shall use DAS protocol sequence response format
FR4	mycoCLAP DAS DNA/protein reference servers shall be able to provide a response based on the start and end points provided by the user

Table 5: Functional requirements of the mycoCLAP DAS DNA/protein reference servers

ID	**Description**
NFR1	mycoCLAP DAS DNA/protein reference servers shall get the data from mycoCLAP database which is made in MySQL and follows BioXRT format
NFR2	mycoCLAP DAS DNA/protein servers shall be written in Java
NFR3	mycoCLAP DAS DNA/protein servers shall use Dazzle APIs
NFR4	mycoCLAP DAS DNA/protein reference servers shall allow multiple users to access the system simultaneously
NFR5	mycoCLAP DAS DNA/protein reference servers shall not require user to install special software to use the system
NFR6	The query format of mycoCLAP DAS DNA/protein reference servers shall follow DAS protocol format to query a DAS reference server
NFR7	The XML response of mycoCLAP DAS DNA/protein reference servers shall follow DAS protocol response format for reference servers

Table 6: Non-functional requirements of the mycoCLAP DAS DNA/protein reference servers

5.2 Architecture of the mycoCLAP DAS Reference Servers

The mycoCLAP DAS reference servers are based on the DAS protocol that has similar architecture to Service Oriented Architecture (SOA). As mentioned in Chapter 2, Section 2.2.2.1, the DAS is an HTTP-based client-server system that uses a layered architecture to support the reuse of "off-the-shelf" software components. The DAS architecture consists of three layers: *the service interface layer, the business logic layer, and the data access layer.*

The *service interface layer* is responsible for providing facilities that allow the end user to communicate with the system. All essential logical functions are performed by the *business logic layer.* This layer also applies business rules to the messages received by the service layer, and isolates these rules from the service interface layer. Finally, the *data access layer* facilitates access to the data [Sko06].

Each layer can be implemented in different computer languages/platforms, and the final system can be a combination of different technologies. For example, a DAS reference server can be written in Java or Perl. This gives developers the freedom to choose an appropriate method of implementation.

Generally, there are four implementations of the DAS service interface layer, namely Dazzle and MyDAS in Java, and Proserver and LDAS in Perl. In addition, in the data access layer, different types of data storage can be used such as relational databases such as MySQL or SQL server database, or different text-based formats can be used, such as GFF data files. All the developers mainly need to know is, if the message between the server and the clients follows the DAS protocol format.

Figure 28 exhibits (in greater detail) how these layers and software components work with each other. These components can be mapped to modules that are found in the mycoCLAP DAS reference servers.

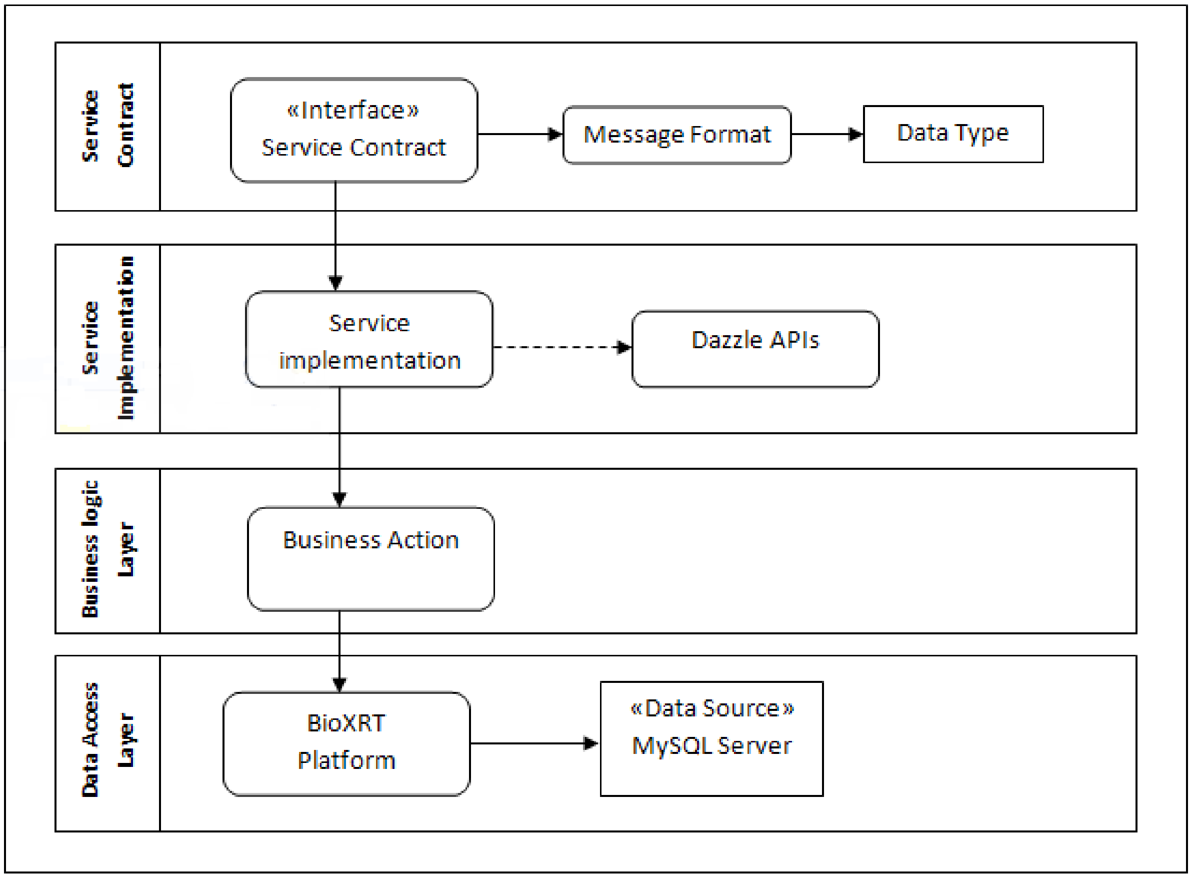


Figure 28: Detail of architecture layers in the mycoCLAP DAS reference servers (modified from SOA architecture [Sko06])

5.2.1 Deployment Diagram for mycoCLAP Reference Servers

The mycoCLAP DAS DNA/protein reference servers work in a client-server paradigm, and can be deployed on several machines. The system deployment diagram, shown in Figure 29, represents how each software artifact is deployed to a server and also how these artifacts communicate with each other. The main software artifact, namely "das.war" is a web application that is responsible for all the functionality of the whole system. It uses the Dazzle library [Dow] to implement a service based on the DAS protocol. The mentioned artifacts are deployed on a web server, allowing the user to access them with a web browser software. The mycoCLAP database is deployed on a database server.

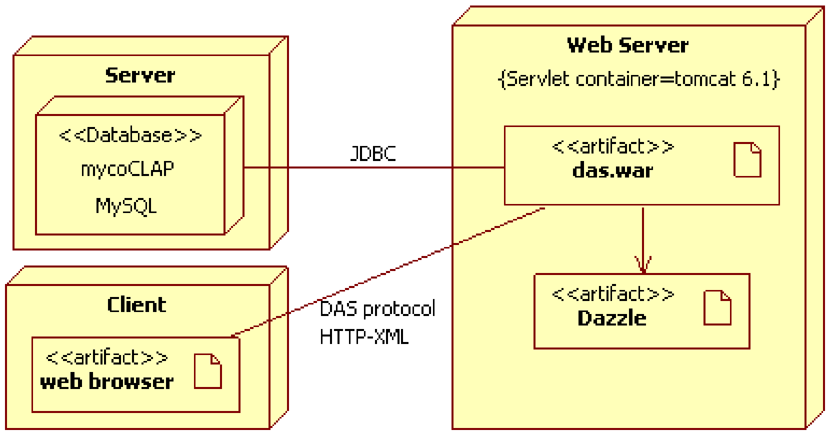


Figure 29: Deployment diagram for mycoCLAP DAS DNA/protein reference servers

5.3 System Design

This section describes the detailed design of the mycoCLAP DAS DNA/protein reference servers system. An overview is provided in the form of an explanation of the major classes in the system, followed by an explanation of the database design.

Based on the DAS protocol, a DAS reference server should respond to a "sequence" command when a sequence is requested by a user. As a result, the mycoCLAP DAS DNA/protein reference servers use the "Dazzle" library [Dow] to provide the sequence command. The server requires the implementation of a special method named "getSequence" of the"DazzleReferenceSource" interface, which can be found in the "datasource" package of Dazzle.

Figure 30 shows the main classes of the system that are involved in the process of responding to users' requests. In this figure, the two main classes which created the DNA and protein reference servers, namely, "FungiDataSourceDNA" and "FungiDataSourceProtein" have implemented "DazzleReferenceSource" interface and its "getSequence" method.

The "mycoClapSeqFetcher " class is responsible for connecting to the database and fetching the requested DNA or protein sequences. This class calls the "FungiDataSourceDNA" class if the request is for a DNA sequence, or it calls the"FungiDataSourceProtein"

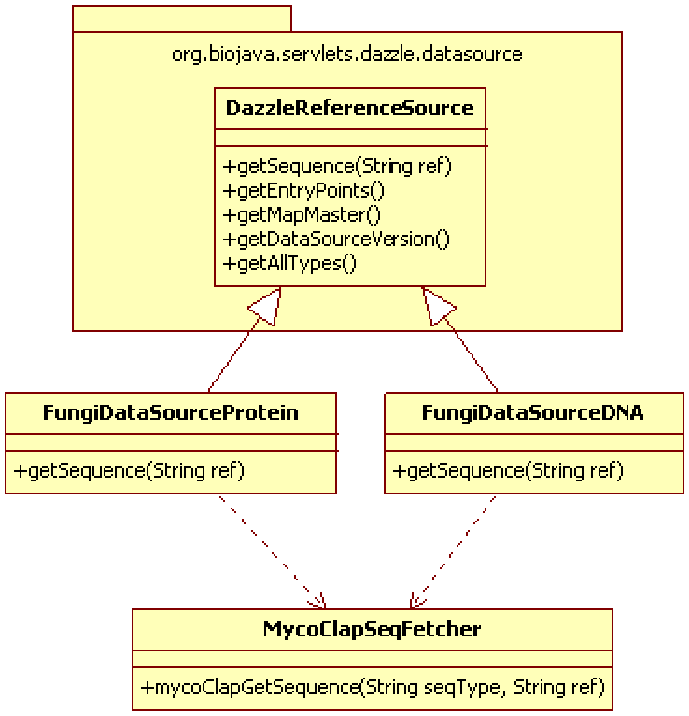


Figure 30: Class diagram of mycoCLAP DAS DNA/protein reference servers

class if the request is for a protein sequence.

5.4 Database Design

The "mycoCLAP" database is designed based on the "BioXRT", which is a platform for developing biological databases [ZFD+06] and has been implemented by the "MySQL" database management system.

Figure 31 shows the data structure of the mycoCLAP database in the BioXRT platform. The platform virtually keeps all tables, attributes and data in four cross-referenced tables. Three of the tables are described as follows:

The"gclass" table keeps the name of all the mycoCLAP tables, "gattribute" table contains all of the attributes for all of the mycoCLAP tables and finally "gdata" table keeps all of the data. As a result, reading the data that is stored in a table from a database kept in the BioXRT format such as the mycoCLAP database is a bit complicated. For example, the mycoCLAP DAS reference server program needs to query these three tables several times to finally find the target data. The structure of the tables and the relationships among them are shown in Figure 31.

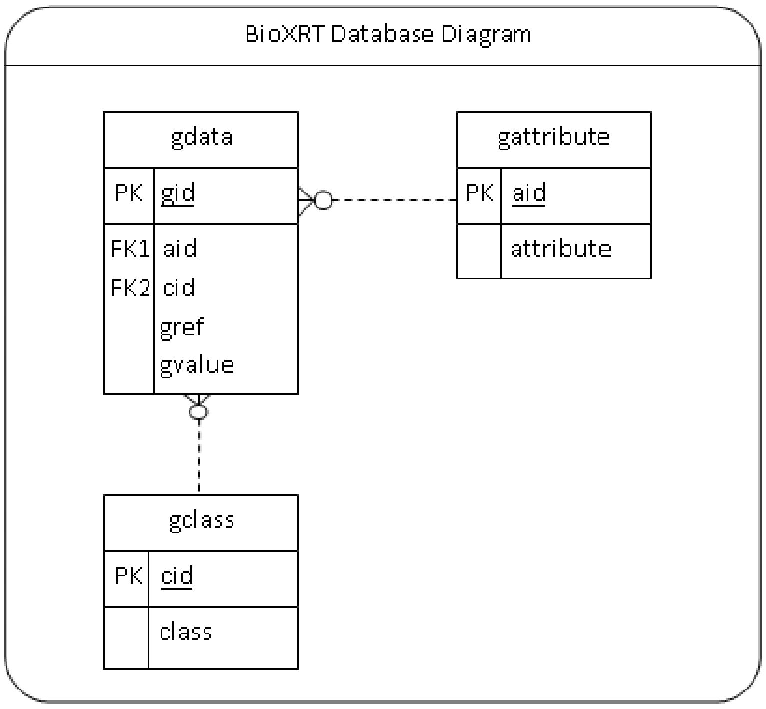


Figure 31: The structure of the three tables in the BioXRT format

Although the mycoCLAP is a large database with several tables, for the purpose of setting up a DAS reference server and fetching the DNA or protein sequences stored in the database, only the "EnzymeSequence" table is needed from the mycoCLAP database.

Figure 32 shows the EnzymeSequence table attributes. This table stores various details related to Enzymes, such as DNA and protein IDs and sequences. Appendix C contains the required code in order to fetch protein or DNA sequences from the mycoCLAP database.

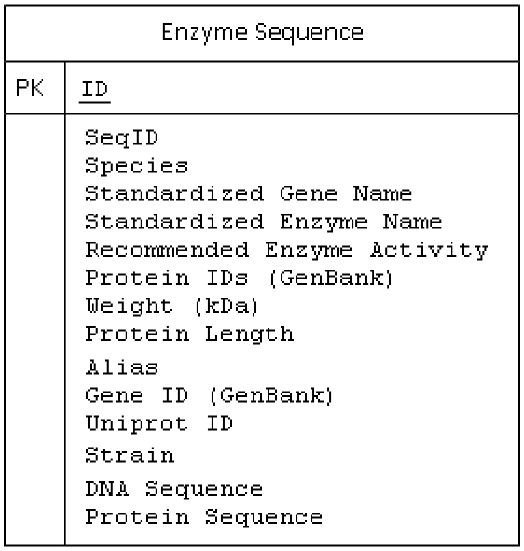
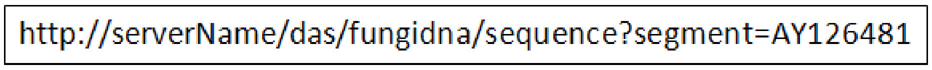


Figure 32: EnzymeSequence Table

5.5 Query Formats and Outputs

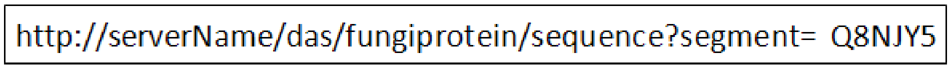
To query the mycoCLAP DAS DNA/protein reference servers, a pre-defined URL is required. The URL query for the DNA server is shown in Figure 33.



```
http://serverName/das/fungidna/sequence?segment=AY126481
```

Figure 33: The URL query for the mycoCLAP DNA DAS reference server

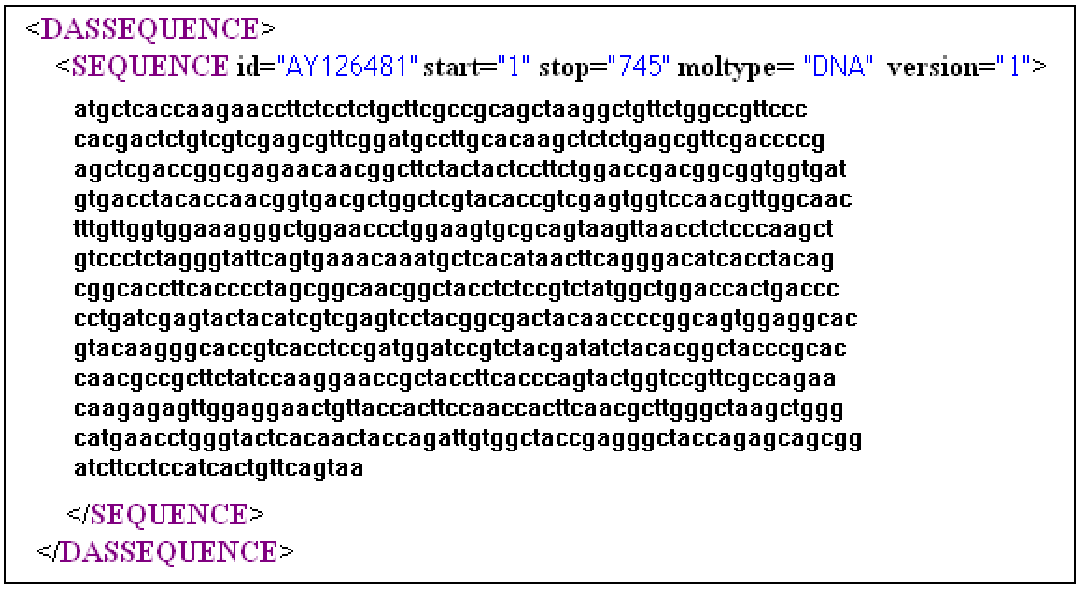
Regarding this query, "fungidna" term specifies the name of the data source from which user fetches the DNA sequences. The "sequence" term in the query indicates the DAS command for fetching a sequence, and finally, the "segment" term specifies the ID for the sequence that a user is supposed to get as a response. The URL query for the protein server is similar to the URL query for the DNA server, except the data source name is "fungiprotein" instead of "fungidna". The URL query for the protein server is shown in the Figure 34.



```
http://serverName/das/fungiprotein/sequence?segment= Q8NJY5
```

Figure 34: The URL query for the mycoCLAP protein DAS reference server

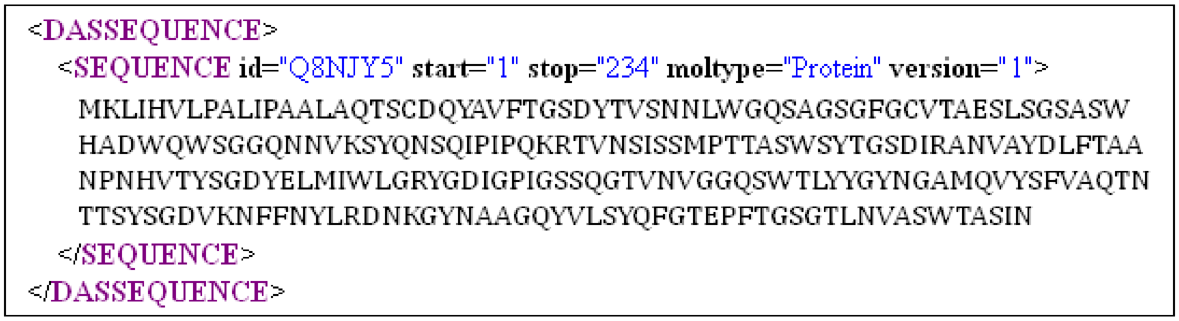
The response for querying the DNA DAS reference server for the above sequence ID is shown below in Figure 35.



```
<DASSEQUENCE>
  <SEQUENCE id="AY126481" start="1" stop="745" moltype= "DNA"  version="1">
    atgctcaccaagaaccttctcctctgcttcgccgcagctaaggctgttctggccgttccc
    cacgactctgtcgtcgagcgttcggatgccttgcacaagctctctgagcgttcgaccccg
    agctcgaccggcgagaacaacggcttctactactccttctggaccgacggcggtggtgat
    gtgacctacaccaacggtgacgctggctcgtacaccgtcgagtggtccaacgttggcaac
    tttgttggtggaaagggctggaaccctggaagtgcgcagtaagttaacctctcccaagct
    gtccctctagggtattcagtgaaacaaatgctcacataacttcagggacatcacctacag
    cggcaccttcaccccctagcggcaacggctacctctccgtctatggctggaccactgaccc
    cctgatcgagtactacatcgtcgagtcctacggcgactacaaccccggcagtggaggcac
    gtacaagggcaccgtcacctccgatggatccgtctacgatatctacacggctacccgcac
    caacgccgcttctatccaaggaaccgctaccttcacccagtactggtccgttcgccagaa
    caagagagttggaggaactgttaccacttccaaccacttcaacgcttgggctaagctggg
    catgaacctgggtactcacaactaccagattgtggctaccgagggctaccagagcagcgg
    atcttcctccatcactgttcagtaa
  </SEQUENCE>
</DASSEQUENCE>
```

Figure 35: The response of "fungidna" DAS reference server for sequence ID "AY126481"

Another example is the response for querying the"fungiprotein" data source for sequence ID "Q8NJY5", which is shown in Figure 36.



```
<DASSEQUENCE>
  <SEQUENCE id="Q8NJY5" start="1" stop="234" moltype="Protein" version="1">
    MKLIHVLPALIPAALAQTSCDQYAVFTGSDYTVSNNLWGQSAGSGFGCVTAESLSGSASW
    HADWQWSGGQNNVKSYQNSQIPIPQKRTVNSISSMPTTASWSYTGSDIRANVAYDLFTAA
    NPNHVTYSGDYELMIWLGRYGDIGPIGSSQGTVNVGGQSWTLYYGYNGAMQVYSFVAQTN
    TTSYSGDVKNFFNYLRDNKGYNAAGQYVLSYQFGTEPFTGSGTLNVASWTASIN
  </SEQUENCE>
</DASSEQUENCE>
```

Figure 36: The response of "fungiprotein" DAS reference server for sequence ID "Q8NJY5"

Chapter 6

Case study

In this chapter, in order to provide an overview of how the DasSeqFetcher tool and the Mobyle portal work together to help scientists complete a study, a phylogenetic (defined as evolutionary-informative branch lengths) tree study is explained. A phylogeny tree is a branching diagram that shows the evolutionary relationship between sequences with regards to the similarities and differences between the predicted functional groups [AD09]. Figure 43, below, shows a phylogeny tree [FM67].

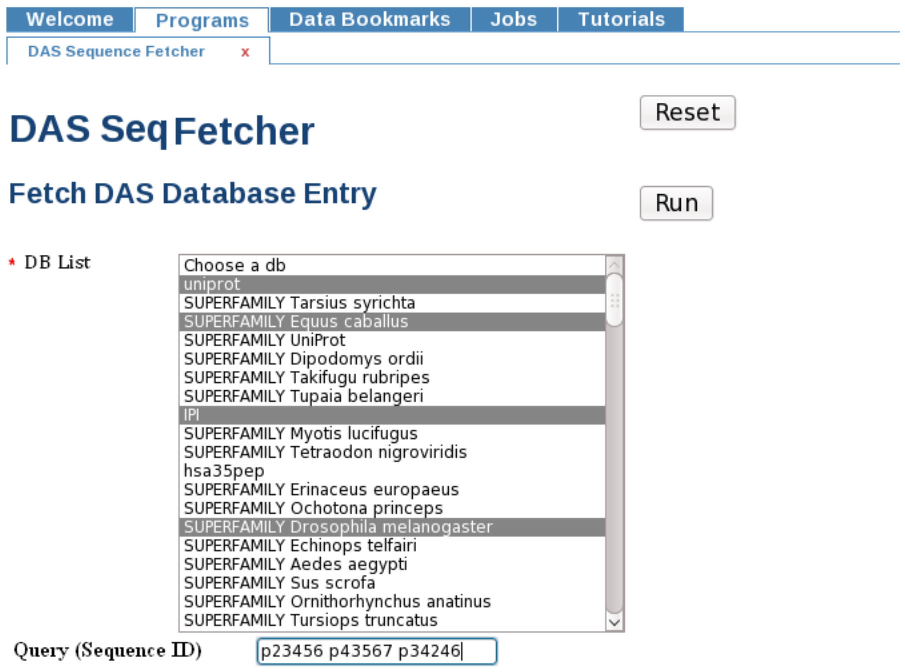
A challenging problem in annotating data in bioinformatics refers to classifying coherent sequences into functional groups or classifying a sequence within the most related functional group. One of the methods that is used in this regard is called a "profile to sequence" comparison (such as an HMM profile), which can also be used to cluster a homologous sequence in a functional group.

OrthoMCL [LSR03] is a sequence clustering program that groups sequences based on their sequence similarities. In this study, bioinformaticians attempt to analyze the OrthoMCL result in order to evaluate how accurate the output of the software would be, and how close this output is to the real data.

After getting the generated functional clusters from the OrthoMCL program, we feed the sequences of a family of functional groups into a phylogenetic tree generation program in order to cross validate the generated functional groups. Phylogenetic trees are not benchmarks for functional groups. However, they can be used along with confidence levels of the branches as cross validation methods.

In order to prove that the DasSeqFetcher has the ability to work with Mobyle, the focus of the current case study is to make a phylogenetic tree based on the OrthoMCL program's output by using the DasSeqFetcher to import the required data into the Mobyle portal.

To begin the process of producing the phylogenetic tree, the required sequences need to be loaded to the Mobyle portal so that they can be used for the next analysis. The extracted sequence IDs from the OrthoMCL output have been entered to the query text box of the DasSeqFetcher tool. Then those databases that can provide the sequences have been selected by the user from the database list, as in Figure 37, below.



DAS Seq Fetcher

Fetch DAS Database Entry

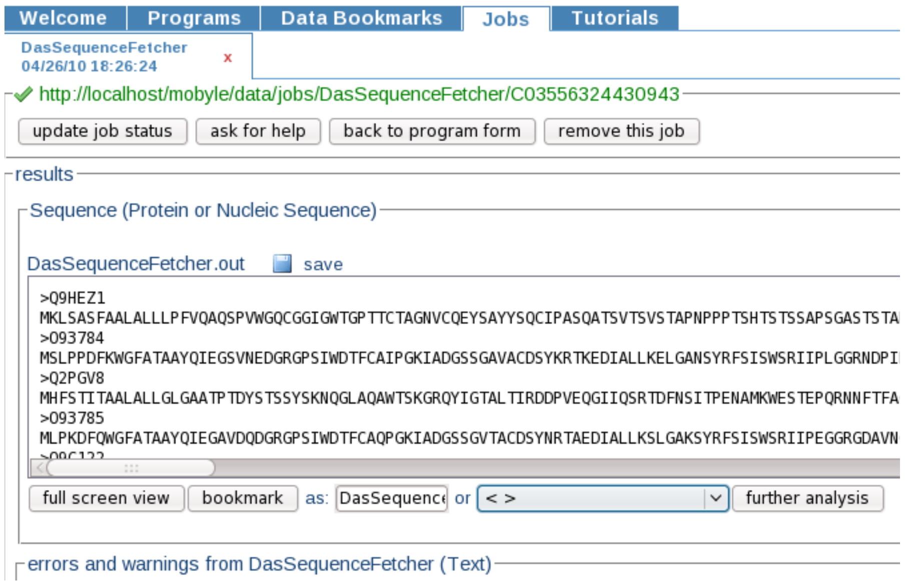
* DB List

Choose a db
uniprot
SUPERFAMILY Tarsius syrichta
SUPERFAMILY Equus caballus
SUPERFAMILY UniProt
SUPERFAMILY Dipodomys ordii
SUPERFAMILY Takifugu rubripes
SUPERFAMILY Tupaia belangeri
IPI
SUPERFAMILY Myotis lucifugus
SUPERFAMILY Tetraodon nigroviridis
hsa35pep
SUPERFAMILY Erinaceus europaeus
SUPERFAMILY Ochotona princeps
SUPERFAMILY Drosophila melanogaster
SUPERFAMILY Echinops telfairi
SUPERFAMILY Aedes aegypti
SUPERFAMILY Sus scrofa
SUPERFAMILY Ornithorhynchus anatinus
SUPERFAMILY Tursiops truncatus

Query (Sequence ID) p23456 p43567 p34246

Figure 37: The "DasSeqFetcher" program interface

The DasSeqFetcher returns those sequences that their IDs have been previously entered in the “Query” text box. The DasSeqFetcher output for a list of sequences is shown in Figure 38. Although the results are shown in a FASTA format, the “squizz_convert” tool is used to change them to a standard FASTA format.



```
>Q9HEZ1
MKLSASFAALALLLPFVQAQSPVWGQCGGIGWTGPTTCTAGNVCQEYSAYYSQCIPASQATSVTSVSTAPNPPPTSHTSTSSAPSGASTSTA
>O93784
MSLPPDFKWGFATAAYQIEGSVNEDGRGPSIWDTFCAIPGKIADGSSGAVACDSYKRTKEDIALLKELGANSYRFSISWSRIIPLGGRNDPI
>Q2PGV8
MHFSTITAALALLGLGAATPTDYSTSSYSKNQGLAQAWTSKGRQYIGTALTIRDDPVEQGIIQSRTDFNSITPENAMKWESTEPQRNNFTFA
>O93785
MLPKDFQWGFATAAYQIEGAVDQDGRGPSIWDTFCAQPGKIADGSSGVTACDSYNRTAEDIALLKSLGAKSYRFSISWSRIIPEGGRGDAVN
```

Figure 38: The sequence retrieved by “DasSeqFetcher” program

To make a phylogeny tree, four steps are required that are described as follows:

1. A distance matrix is made, based on the Multiple Sequences Alignment (MSA) data.

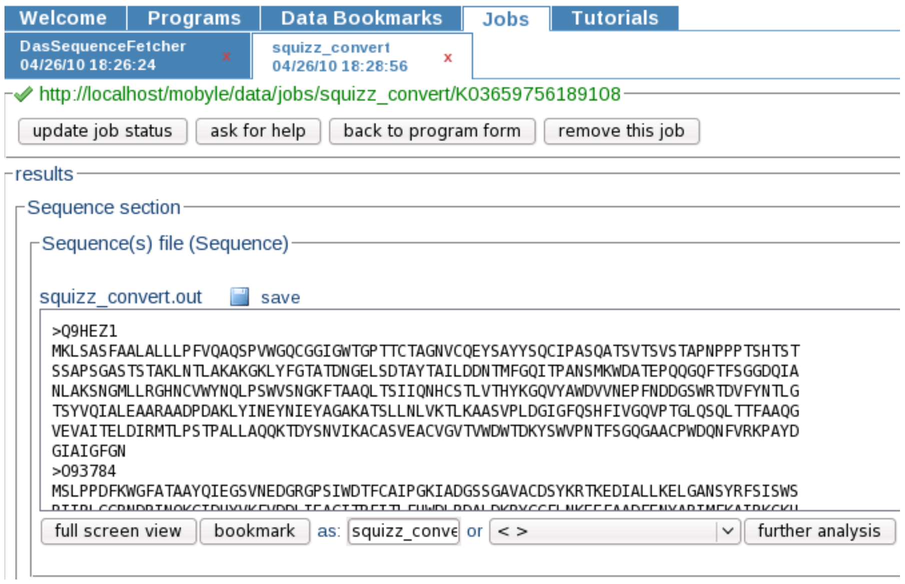
2. By using this matrix, a simple tree is created.

3. The simple tree is used as the basic input to make an optimized tree.

4. By using a phylogenetic tree drawing program, the final tree is drawn.

For example, in Figure 39, “ClustalW”, which is an MSA program, is selected by the user from the list box that is available under the “squizz_convert” result box, and then the “further analysis” button is selected by the user.

The next window shows the ClustalW program interface with the FASTA sequences inside its input box. The default parameters can be accepted and some can

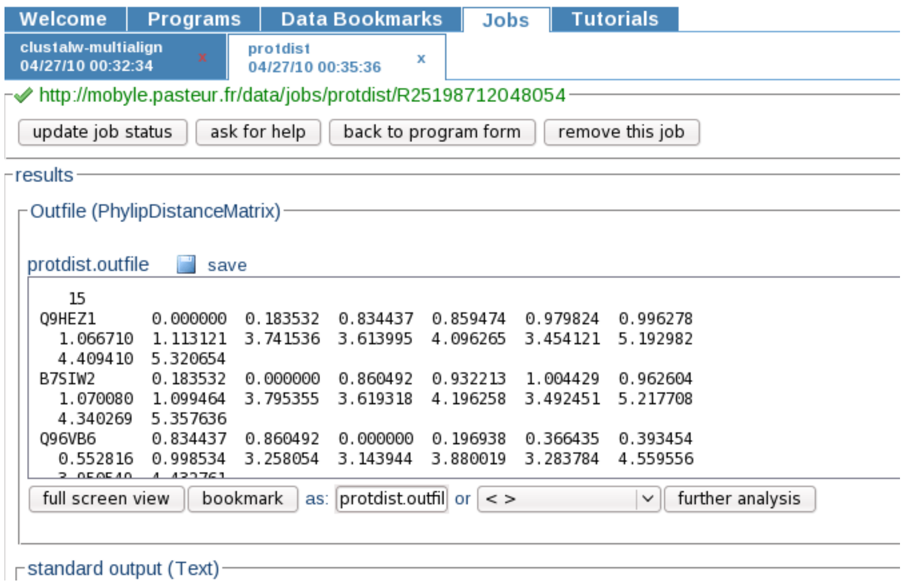


```
>Q9HEZ1
MKLSASFAALALLLPFVQAQSPVWGQCGGIGWTGPTTCTAGNVCQEYSAYYSQCIPASQATSVTSVSTAPNPPPTSHTST
SSAPSGASTSTAKLNTLAKAKGKLYFGTATDNGELSDTAYTAILDDNTMFGQITPANSMKWDATEPQQGQFTFSGGDQIA
NLAKSNGMLLRGHNCVWYNQLPSWVSNGKFTAAQLTSIIQNHCSTLVTHYKGQVYAWDVVNEPFNDDGSWRTDVFYNTLG
TSYVQIALEAARAADPDAKLYINEYNIEYAGAKATSLLNLVKTLKAASVPLDGIGFQSHFIVGQVPTGLQSQLTTFAAQG
VEVAITELDIRMTLPSTPALLAQQKTDYSNVIKACASVEACVGVTVWDWTDKYSWVPNTFSGQGAACPWDQNFVRKPAYD
GIAIGFGN
>O93784
MSLPPDFKWGFATAAYQIEGSVNEDGRGPSIWDTFCAIPGKIADGSSGAVACDSYKRTKEDIALLKELGANSYRFSISWS
```

Figure 39: Squizz_convert output in FASTA format

be changed, based on a user's needs and then the ClustalW program is run. The program output is the aligned sequence information, which is ready to use as input in order to generate a "distance matrix". For example, a program named "Protdist" from the "PHYLIP" package is responsible for generating a "distance matrix". A distance matrix is a matrix that contains a calculated distance between each pair of sequences. It assigns a specific score for each match, mismatch and gap between two sequences and finally calculates the distance score for each pair. This score shows the evolutionary distance between the two sequences [Mou04]. Figure 40 shows the distance matrix made by the "Protdist" program.

Based on the above information, the phylogeny tree is made. The distance scores affect the final length of the branches between the two sequences in the tree. The tree is made using the "Neighbor-Joining" program [SN87], which uses a bottom-up approach to make the tree. It uses scores from the distance matrix to pair two sequences that have the smallest distance between one another and considers these two sequences as one sequence or node in the tree. Then, new scores are calculated for all sequences, taking into consideration the newly added node. This process continues



```
    15
Q9HEZ1      0.000000  0.183532  0.834437  0.859474  0.979824  0.996278
  1.066710  1.113121  3.741536  3.613995  4.096265  3.454121  5.192982
  4.409410  5.320654
B7SIW2      0.183532  0.000000  0.860492  0.932213  1.004429  0.962604
  1.070080  1.099464  3.795355  3.619318  4.196258  3.492451  5.217708
  4.340269  5.357636
Q96VB6      0.834437  0.860492  0.000000  0.196938  0.366435  0.393454
  0.552816  0.998534  3.258054  3.143944  3.880019  3.283784  4.559556
```

Figure 40: Protdist result (a distance matrix)

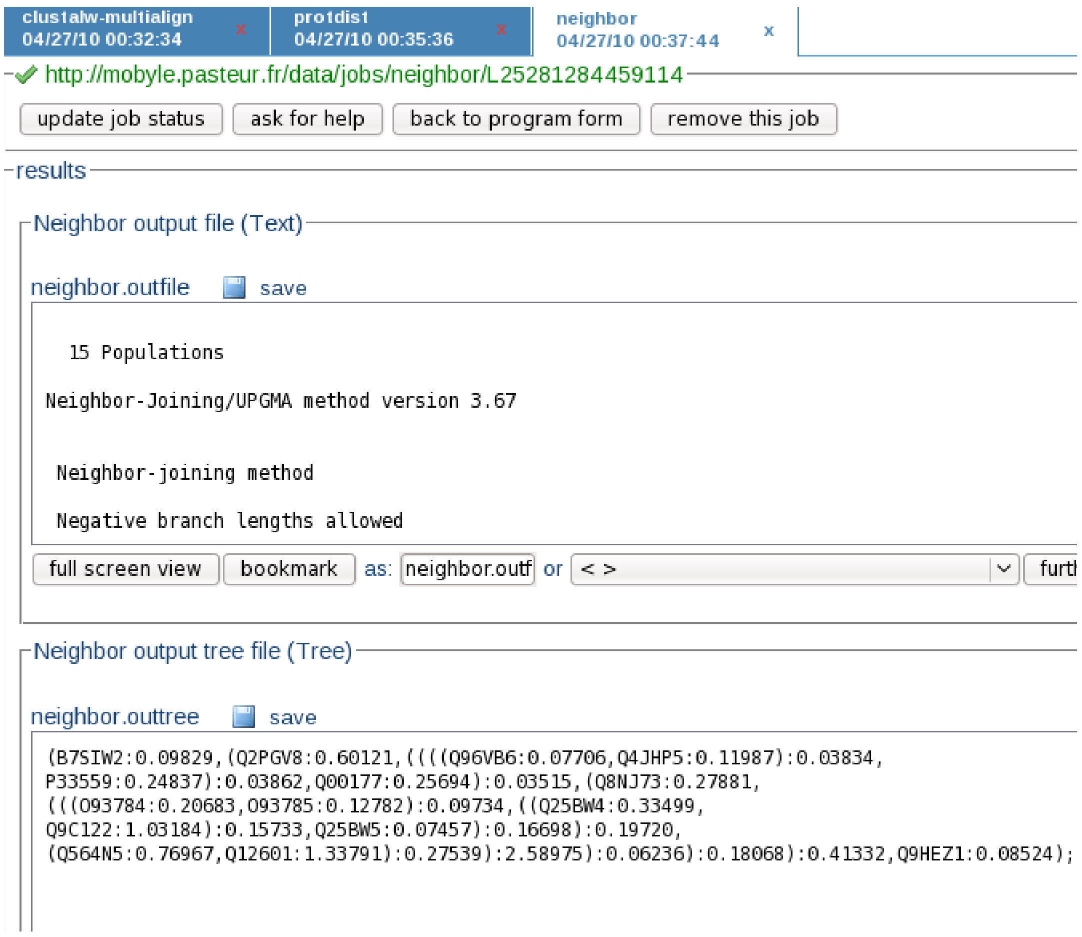
until all sequences find a sequence or subtree neighbor [GS06]. Figure 41 shows the basic tree that is made by the Neighbor-Joining program.

This tree can be optimized using the “Phyml ” program to provide a more accurate result. The Phyml program uses a hill-climbing algorithm to find a more optimized tree among the possibly available trees.

Hill-climbing algorithms attempt to improve the answer gradually, in several iterations. The algorithm ends when no further improvements are possible. With this approach, the final answer may not be the best, but it is much better than the non-elaborated primary tree and is made with a quick, distance-based algorithm [GG03].

The inputs for Phyml program include the “Neighbor-Joining” output tree, which is placed in the “starting tree” text box, and secondly, the MSA result that is entered in the “sequence alignment” textbox. The data type parameter is selected based on the sequence type, that can be DNA or Amino acids as it is shown in Figure 42, below.

Finally, the “newicktops” program is used to draw the tree in a user-friendly way, which shows the branches and the length of each branch based on the lineage time.

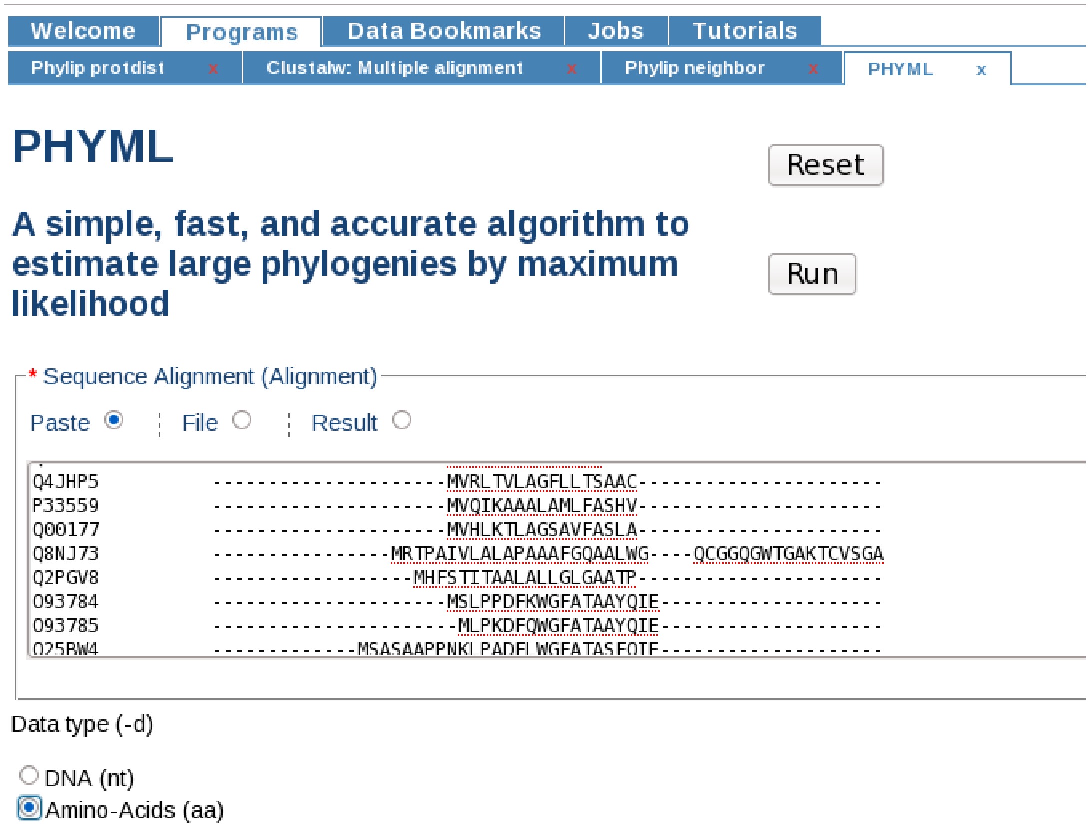


```
  15 Populations

Neighbor-Joining/UPGMA method version 3.67


 Neighbor-joining method

 Negative branch lengths allowed
```

```
(B7SIW2:0.09829,(Q2PGV8:0.60121,((((Q96VB6:0.07706,Q4JHP5:0.11987):0.03834,
P33559:0.24837):0.03862,Q00177:0.25694):0.03515,(Q8NJ73:0.27881,
(((O93784:0.20683,O93785:0.12782):0.09734,((Q25BW4:0.33499,
Q9C122:1.03184):0.15733,Q25BW5:0.07457):0.16698):0.19720,
(Q564N5:0.76967,Q12601:1.33791):0.27539):2.58975):0.06236):0.18068):0.41332,Q9HEZ1:0.08524);
```

Figure 41: Basic tree made by the Neighbor-Joining program

The final tree is shown in Figure 43.



PHYML

A simple, fast, and accurate algorithm to estimate large phylogenies by maximum likelihood

* Sequence Alignment (Alignment)

[✓ Paste] File Result

```
Q4JHP5          ---------------------MVRLTVLAGFLLTSAAC----------------------
P33559          ---------------------MVQIKAAALAMLFASHV----------------------
Q00177          ---------------------MVHLKTLAGSAVFASLA----------------------
Q8NJ73          ----------------MRTPAIVLALAPAAAFGQAALWG----QCGGQGWTGAKTCVSGA
Q2PGV8          ------------------MHFSTITAALALLGLGAATP----------------------
O93784          ---------------------MSLPPDFKWGFATAAYQIE--------------------
O93785          ----------------------MLPKDFQWGFATAAYQIE--------------------
O25BW4          -------------MSASAAPPNKLPADFLWGFATASFQIE--------------------
```

Data type (-d)

DNA (nt) [✓ Amino-Acids (aa)]

Figure 42: Phyml program interface and its input parameters

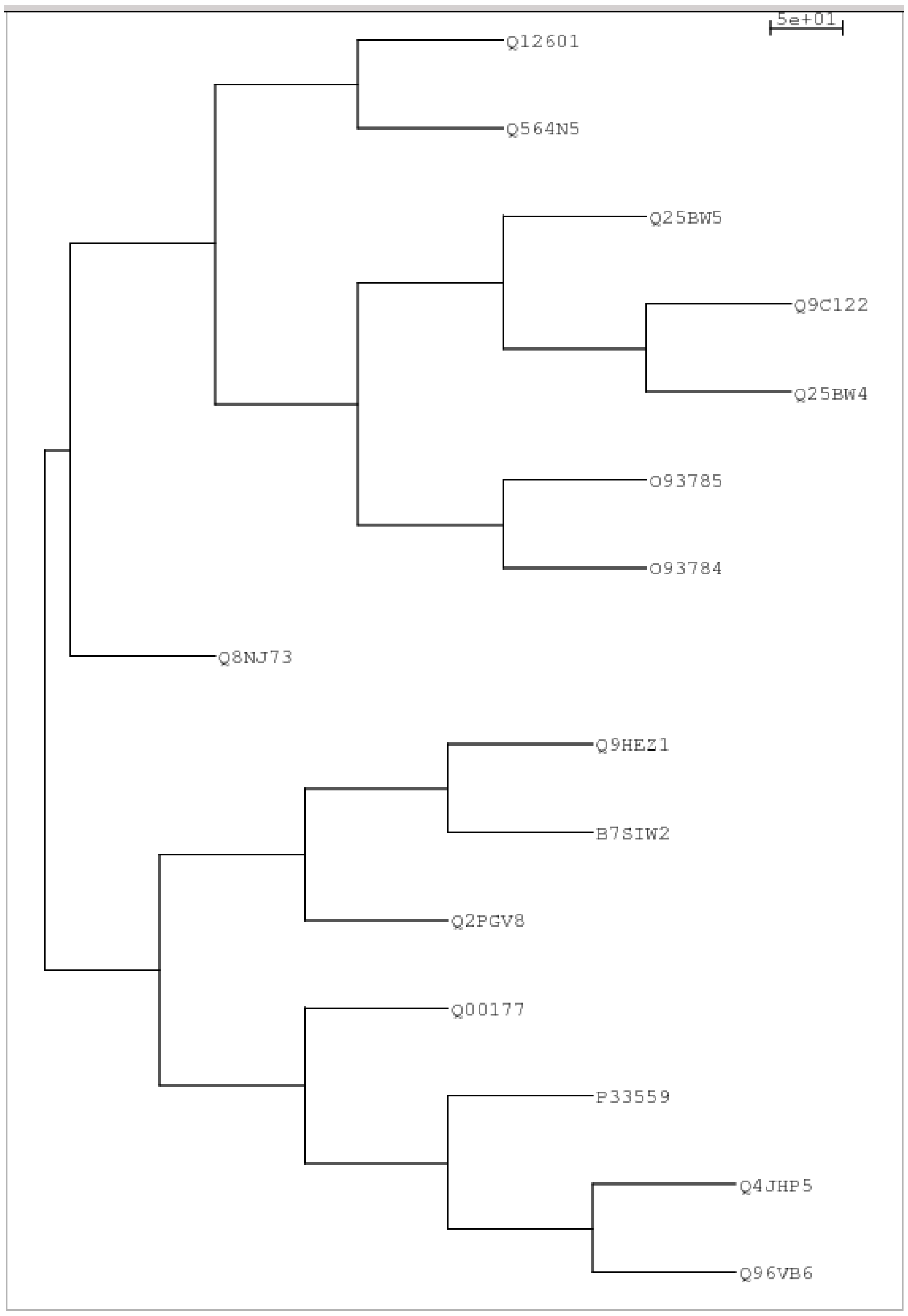


Figure 43: Newicktops output tree, the phylogenetic tree based on the input sequences

Chapter 7

Conclusion

Biological data are by nature, highly heterogeneous. Additionally, the availability of computational tools as a result of new developments in computer hardware has led to a significant increase in the amount of biological data. Furthermore, data are generated with different formats by isolated research groups. On the other hand, biologists use a wide variety of analysis tools, each of which has its own input/output data format and installation and upgrading process. As a result, the integration of data from heterogeneous sources and the use of different tools to process these data is a cumbersome and tedious task.

As the CBN research group has had the same difficulties regarding data and tool integration, upon reviewing the available bioinformatics tools and data integration methods, the DAS protocol was used as a data integration method. The DAS protocol was chosen because it offers a standard data format, a communication method, and a query format for exchanging biological data. Consequently, two DAS reference servers for DNA and protein sequences were developed, in order to provide the data that was stored in the mycoCLAP database. Also, the Mobyle portal was selected as a tool integration system. Mobyle was found to be a convenient and efficient tool by the research groups, mainly as a result of the ease with which the desirable software was installed and its user-friendly environment.

The main problem regarding the Mobyle portal is that although Mobyle has two data aggregator tools, Golden and SeqretFromUSA, these tools are not DAS protocol compatible and they cannot connect to the mycoCLAP DAS reference servers nor to the other available DAS reference servers. They can only fetch sequences from well-known databases.

To solve this problem, it was necessary to develop a new tool that connects the Mobyle portal to the growing number of biological data provided by DAS servers. This tool would also meet the users' requirements of a data aggregator tool. In addition, the tool had to work based on the DAS protocol and had to be compatible with the Mobyle portal.

Regarding the main contribution of the present thesis, an attempt was made to develop a tool, named, DasSeqFetcher. This tool provides an up-to-date list of operating DAS reference servers, sorted by their nicknames. It automatically updates its own DAS server list, based on new information in the DAS server registry. Using this new tool, users do not need to collect data from different data sources; instead, the tool collects sequences from several DAS reference servers simultaneously and merges them into an output. Users can send the server names and sequence IDs as input parameters to the tool, and then the tool returns sequences to the users in the FASTA format. Also, the DasSeqFetcher checks the functionality of the DAS reference servers that users try to connect with. Moreover, in case the first DAS server in the user's server list does not provide the data, the DasSeqFetcher queries an alternative DAS reference server in the list in order to find the data. Therefore, the DasSeqFetcher can address the user requirements and can cover the weaknesses associated with Golden and SeqretFromUSA tools.

In order to validate the DasSeqFetcher tool, several tests were performed to evaluate the tool as per the user requirements and to validate the design and usability of the tool. Also, a load test was accomplished to measure the effect of load work on the tool's performance. Results revealed that the tool could help users obtain the required data much faster and more easily than a manual method of data collection and even faster than using a Linux shell script. However, the results also showed that the DAS servers' performance was a bottleneck of the DasSeqFetcher program performance. Also, the number of connections to the servers played an important role in the success of the new tool. To cope with the limitation of the number of connections, the DasSeqFetcher design was improved by sending a constant and optimized number of requests simultaneously. Even though this approach caused the fetch process to be longer, in time, it ensured that the DasSeqFetcher could be scaled up to serve thousands of simultaneous requests, even if the number of sequence IDs was increased over the server's limitations.

One major lesson learned from this study is that the performance of a new program, when written in a distributed environment, depends not only on the program's quality but also on the service quality of the server with which the program communicates. This situation is mainly a result of the nature of such systems that are offered by different data providers, which is out of the programmer's control.

Also, two DAS reference servers were developed in order to publish the data produced by the CBN group. The development of the servers was based on the data kept in mycoCLAP, a database of characterized lignocellulose-active proteins. Registered on the DAS registry, the newly designed "mycoCLAP DAS reference" servers can be accessed in the Mobyle portal by using the DasSeqFetcher program.

7.1 Future Work

Computer-based biological tools work not only with the protein or nucleotide sequences, but also can work with features related to these sequences, such as domains, active sites and so on. the present work could be further developed in the future by a new tool for the Mobyle portal, to equip it with the ability to connect with the DAS annotation servers. As a result, it may possible to import the DAS annotation servers' data such as sequence features, structures and alignment information to the Mobyle portal.

In addition, the DasSeqFetcher can be improved by letting users retrieve only a part of the sequence, instead of the whole sequence. The DasSeqFetcher can also be extended in order to provide an application that fetches the sequences from a server based on the type, method or category of the server. It should be noted that the sequences can be presently accessed only according to their sequence IDs.

Appendix A

DasSeqFetcher.xml file

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- XML Authors: Hajar Sadrarhami -->
<!-- Distributed under LGPLv2 Licence. Please refer to the COPYING.LIB document. -->
<program>
  <head>
    <name>DasSequenceFetcher</name>
    <version>1.0</version>
    <doc>
      <title>DasSeqFetcher</title>
      <description>
        <text lang="en">Fetch DAS Database Entry</text>
      </description>
      <authors>Hajar Sadrarhami</authors>
    </doc>
    <category>database:search:sequence</category>
    <command>DasSequenceFetcher </command>
  </head>
  <parameter ismandatory="1" issimple="1" ismaininput="1">
    <name>db</name>
    <prompt lang="en">Database</prompt>
    <type>
      <datatype>
       <class>MultipleChoice</class>
      </datatype>
    </type>
    <vdef>
      <value>null</value>
    </vdef>
    <xi:include href="http://compute.fungalgenomics.ca /DasServerList/XmlServerList">
      <xi:fallback>
        <vlist>
          <velem undef="1">
            <value>null</value>
            <label> an error accrued </label>
          </velem>
        </vlist>
      </xi:fallback>
    </xi:include>
    <separator>,</separator>
    <format>
        <code proglang="perl">" \$db"</code>
        <code proglang="python">" " + db + " "</code>
    </format>
    <argpos>1</argpos>
   </parameter>
   <parameter ismandatory="1" issimple="1" ismaininput="1">
      <name>query</name>
      <prompt lang="en">Query (Enter accession numbers with a space to separate)</
          prompt>
      <type>
        <datatype>
          <class>String</class>
        </datatype>
      </type>
      <precond>
       <code proglang="perl">$db</code>
       <code proglang="python">db is not None</code>
      </precond>
      <format>
       <code proglang="perl">" $value"</code>
       <code proglang="python">" "+str(value)</code>
      </format>
      <argpos>2</argpos>
      <example>
       P23456 P34567
      </example>
   </parameter>

   <parameter isstdout="1">
      <name>DasSequenceFetcher_out</name>
      <prompt lang="en">Sequence</prompt>
      <type>
        <biotype>Protein</biotype>
        <biotype>Nucleic</biotype>
        <datatype>
          <class>Sequence</class>
        </datatype>
      </type>
      <filenames>
        <code proglang="perl">"DasSequenceFetcher.out"</code>
        <code proglang="python">"DasSequenceFetcher.out"</code>
      </filenames>
    </parameter>
  </parameters>
</program>
```

Appendix B

DasSeqFetcher Installation Instructions on the Mobyle Portal

There are several tools come with the Mobyle package that help to deploy a new program. First of all, the validity of XML interface file should be checked using "mobvalid" which can be found under "tools" folder in the Mobyle folder. The command to call the program is:

```
mobvalid file-name
```

This program checks the correctness of the XML file based on the structure explained in the Mobyle XML format using RelaxNG rules.

Mobyle has two installation versions, an application version known as MOBYLEHOME and the other one is the web application version. The MOBYLEHOME version is a platform for the web version, i.e. all required tools and files which are essential for deploying a new program are in this version. To deploy a program using Mobyle tools, the XML interface file is placed in the "MOBYLEHOME /local/programs" folder in application version and the executable file in the path that is specified in the "mobyle/local/config/config.py" file under BINARY_PATH property. In addition, the path to the application folder should be defined in the system path. Finally, the new program is deployed under the web application version using “mobdeploy” tool.

Web application version is installed under the web directory and Apache is used as its web container. Web Mobyle folder contains portal folder which includes html, applet files and portal web pages. It also contains data folder which keeps interface XML files for all the available programs as well as sessions and jobs information. The software is accessible from the web version. To install a web version of a program, server administrator should call the mobdeploy tool from “MOBYLEHOME /tools/” folder using the following command:

```
mobdeploy -p DasSequenceFetcher -s local deploy
```

The parameter “-p DasSeqenceFetcher” introduces the name of the program to deploy and “-s local” introduces the server on which the program is deployed, which is the local machine in this case. This command deploys the program on the web portal and adds the new program in the Mobyle index. Consequently, it can be seen in the program tree on the left panel of the Mobyle portal.

It should be mentioned that when the DasSeqFetcher program is deployed, the XML interface file reads the data from “dynamic include” and places the server list data as a static part of DasSeqenceFetcher.xml file. As a result, when the program page loads, the dynamic server list will not be shown. To solve this problem and have the data in dynamic manner, “DasSequenceFetcher.xml” file should be copied in the “/www/mobyle/programs” folder after the installation is completed.
In brief, the steps needed to install “DasSequenseFetcher” program are as followed:

1. Creating the interface XML file (DasSequenceFetcher.xml) and put it in the “MOBYLEHOME/Local/Programs”.

2. Validating the XML file with mobvalid tool using the following command:

```
mobvalid MOBYLEHOME/Local/Programs/DasSequenceFetcher.xml
```

3. Making the "DasSequenceFetcher.jar" file followed by copying it in "/www/cgi-bin" web server directory.

4. Creating the "DasSequenceFetcher" batch file and copying in the mobyle BINARY_PATH which can be "/usr/local/bin"

5. Starting web server using the following command to run "DasServerList" program under it:

```
Service httpd start
```

6. Deploying the program using the following command:

```
mobdeploy -p DasSequenceFetcher -s local deploy
```

7. Replacing "DasSequenceFetcher.xml" in the web version ("/www/mobyle/programs") with the same file in the application version directory.

Appendix C

Code to Retrieve DNA and Protein Sequences from mycoCLAP Database

```
package fungiplugins;

import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.NoSuchElementException;

import javax.servlet.ServletConfig;
import javax.servlet.ServletException;

import org.biojava.bio.seq.Sequence;
import org.biojava.servlets.dazzle.datasource.DataSourceException;


public class MycoClapSeqFetcher {

 private String furl,user, pass;

 public MycoClapSeqFetcher(){
        try{
        InputStream configStream = getClass().getResourceAsStream("/dbconfig.txt");
```

```
                BufferedReader br = new BufferedReader(new InputStreamReader(
                    configStream));

                String strLine;
                int i=0;

                while((strLine=br.readLine())!=null){
                        i++;
                switch (i) {
                  case 1:
                          furl=strLine;
                          break;
                  case 2:
                          user=strLine;
                          break;
                  case 3:
                          pass=strLine;
                          break;
                    }
                 }
                br.close();
        }catch (Exception e){//Catch exception if any
           System.err.println("Error: " + e.getMessage());
}
}
public String mycoClapGetSequence(String seqType, String ref) throws
    NoSuchElementException,
        DataSourceException {

        String protseq_aid="",refrence_id="",prot_gene_aid="",tableName_cid="",
            sequence="",query_gene_aid="'";
        String dbUrl = "jdbc:mysql:"+furl;
        String dbClass = "com.mysql.jdbc.Driver";
        String query_uniprot_aid,query_tableName_cid,query_protseq_aidStatment;

        query_uniprot_aid= "select aid from gattribute where attribute='Uniprot ID'";
        query_gene_aid= "select aid from gattribute where attribute='Gene ID (GenBank
            )'";
        query_tableName_cid="select cid from gclass where class='EnzymeSequence'";
        query_protseq_aidStatment="select aid from gattribute where attribute=?";

        try {
                Class.forName("com.mysql.jdbc.Driver");
                Connection con = DriverManager.getConnection (dbUrl,user,pass);
                Statement stmt = con.createStatement();

                ResultSet rs_aid;
                if (seqType== "DNA Sequence"){
                         rs_aid = stmt.executeQuery(query_gene_aid);
                }else{
                         rs_aid = stmt.executeQuery(query_uniprot_aid);
                }

                rs_aid.first();
                prot_gene_aid = rs_aid.getString(1);
                rs_aid.close();
                System.out.println(prot_gene_aid );

          ResultSet rs_cid = stmt.executeQuery(query_tableName_cid);
                rs_cid.first();
                tableName_cid = rs_cid.getString(1);
                rs_cid.close();
                System.out.println(tableName_cid);

                String query_refrenceStatement = "select gref from gdata where aid=?
                    and cid=? and gvalue= ?";
                PreparedStatement query_refrence;
                ResultSet rs_ref;
                query_refrence = con.prepareStatement(query_refrenceStatement);
                query_refrence.setString(1,prot_gene_aid);
                query_refrence.setString(2,tableName_cid);
                query_refrence.setString(3,ref);
                rs_ref = query_refrence.executeQuery();

                if (rs_ref.next()){
                        refrence_id = rs_ref.getString(1);
                        rs_ref.close();
                        System.out.println(refrence_id);


                        PreparedStatement query_protseq_aid;
                        ResultSet rs_protseq_aid;
                        query_protseq_aid=con.prepareStatement(
                            query_protseq_aidStatment);
                        query_protseq_aid.setString(1, seqType);
                        rs_protseq_aid= query_protseq_aid.executeQuery();
                        rs_protseq_aid.first();
                        protseq_aid = rs_protseq_aid.getString(1);
                        rs_protseq_aid.close();
                        System.out.println(protseq_aid);

                        String query_sequenceStatement="select gvalue from gdata
                            where aid=? and cid=? and gref=?";
                        PreparedStatement query_sequence;
                        ResultSet rs_sequence;
                        query_sequence= con.prepareStatement(query_sequenceStatement)
                            ;
                        query_sequence.setString(1, protseq_aid);
                        query_sequence.setString(2, tableName_cid);
                        query_sequence.setString(3, refrence_id);
                        rs_sequence=query_sequence.executeQuery();

                        rs_sequence.first();
                        sequence = rs_sequence.getString(1);
                        rs_sequence.close();
                        System.out.println(sequence);

                }//end if

                con.close();
        } //end try

        catch(ClassNotFoundException e) {
                e.printStackTrace();
        }

        catch(SQLException e) {
                e.printStackTrace();
        }
        return sequence;
 }
}
```

Bibliography

[ABJ+04] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, and S. Mock. Kepler: An extensible system for design and execution of scientific workflows. In *Scientific and Statistical Database Management, 2004. Proceedings. 16th International Conference on*, pages 423–424. IEEE, 2004.

[AD09] A.M. Altenhoff and C. Dessimoz. Phylogenetic and functional assessment of orthologs inference projects and methods. *PLoS Comput Biol*, 5(1):e1000262, 2009.

[AN05] J. Arlow and I. Neustadt. *UML 2 and the Unified Process: Practical Object-oriented Analysis and Design, 2/e.* Pearson Education India, 2005.

[BBC+02] A. Bateman, E. Birney, L. Cerruti, R. Durbin, L. Etwiller, S.R. Eddy, S. Griffiths-Jones, K.L. Howe, M. Marshall, and E.L.L. Sonnhammer. The Pfam protein families database. *Nucleic acids research*, 30(1):276–280, 2002.

[Bio10] BioDAS. Main page biodas. `http://www.biodas.org/w/index.php?title=Main_Page&oldid=4195`, 2010. [Online; accessed 4-February-2010].

[CA07] S. Cattley and J.W. Arthur. BioManager: the use of a bioinformatics web application as a teaching tool in undergraduate bioinformatics training. *Briefings in bioinformatics*, 8(6):457–465, 2007.

[CGVT03] C. Chothia, J. Gough, C. Vogel, and S.A. Teichmann. Evolution of the protein repertoire. *Science*, 300(5626):1701, 2003.

[CRP05] C. Calero, J. Ruiz, and M. Piattini. Classifying web metrics using the web quality model. *Online Information Review*, 29(3):227–248, 2005.

[DJD$^+$01] R.D. Dowell, R.M. Jokerst, A. Day, S.R. Eddy, and L. Stein. The distributed annotation system. *BMC bioinformatics*, 2(1):2–7, 2001.

[DMK$^+$05] S. Durinck, Y. Moreau, A. Kasprzyk, S. Davis, B. De Moor, A. Brazma, and W. Huber. BioMart and Bioconductor: a powerful link between biological databases and microarray data analysis. *Bioinformatics*, 21(16):3439–3440, 2005.

[Dow] T. Down. Biojava-Dazzel. `http://www.biojava.org/wiki/Dazzle`. [Online; accessed 17-April -2010].

[Dow01] R. Dowell. A distributed annotation system. Master's thesis, Washington University, Computer Science Department, may 2001.

[EUA96] T. Etzold, A. Ulyanov, and P. Argos. [8] SRS: Information retrieval system for molecular biology data banks. *Methods in enzymology*, 266:114–128, 1996.

[Fie00] R.T. Fielding. *Architectural styles and the design of network-based software architectures.* PhD thesis, UNIVERSITY OF CALIFORNIA,IRVINE, 2000.

[FM67] W.M. Fitch and E. Margoliash. Construction of phylogenetic trees. *Science*, 155(760):279–284, 1967.

[FSJ+07] R.D. Finn, J.W. Stalker, D.K. Jackson, E. Kulesha, J. Clements, and R. Pettett. ProServer: a simple, extensible Perl DAS server. *Bioinformatics*, 23(12):1568–1570, 2007.

[GG03] S. Guindon and O. Gascuel. A simple, fast, and accurate algorithm to estimate large phylogenies by maximum likelihood. *Systematic biology*, 52(5):696–704, 2003.

[GRH+05] B. Giardine, C. Riemer, R.C. Hardison, R. Burhans, L. Elnitski, P. Shah, Y. Zhang, D. Blankenberg, I. Albert, J. Taylor, et al. Galaxy: a platform for interactive large-scale genome analysis. *Genome research*, 15(10):1451–1455, 2005.

[GS06] O. Gascuel and M. Steel. Neighbor-joining revealed. *Molecular Biology and Evolution*, 23(11):1997–2000, 2006.

[GS08] C. Goble and R. Stevens. State of the nation in data integration for bioinformatics. *Journal of biomedical informatics*, 41(5):687–693, 2008.

[GW06] B.M. Good and M.D. Wilkinson. The life sciences semantic web is full of creeps! *Briefings in bioinformatics*, 7(3):275–286, 2006.

[HAJvRH03] R.L. Howard, E. Abotsi, E.L. Jansen van Rensburg, and S. Howard. Review-Lignocellulose biotechnology: issues of bioconversion and enzyme production. *African Journal of Biotechnology*, 2(12):602–619, 2003.

[JAB+08] A. Jenkinson, M. Albrecht, E. Birney, H. Blankenburg, T. Down, R. Finn, H. Hermjakob, T. Hubbard, R. Jimenez, P. Jones, et al.

Integrating biological data–the distributed annotation system. *BMC bioinformatics*, 9(Suppl 8):S3, 2008.

[JVD$^+$05] P. Jones, N. Vinod, T. Down, A. Hackmann, A. Kahari, E. Kretschmann, A. Quinn, D. Wieser, H. Hermjakob, and R. Apweiler. Dasty and UniProt DAS: a perfect pair for protein feature visualization. *Bioinformatics*, 21(14):3198–3199, 2005.

[LA04] C. Letondal and O. Amanatian. Participatory design of pipeline tools and web services in bioinformatics. In *Proceedings of Requirements Capture for Collaboration in eScience Workshop, NESC, Edinburgh*, 2004.

[LBW$^+$04] P. Lord, S. Bechhofer, M.D. Wilkinson, G. Schiltz, D. Gessler, D. Hull, C. Goble, and L. Stein. Applying semantic web services to bioinformatics: Experiences gained, lessons learnt. *The Semantic Web–ISWC 2004*, pages 350–364, 2004.

[Let01] C. Letondal. A Web interface generator for molecular biology programs in Unix. *Bioinformatics*, 17(1):73–82, 2001.

[LNE06] U. Leser, F. Naumann, and B. Eckman. *Data Integration in the Life Sciences:third international workshop.* Springer, 2006.

[LPW$^+$06] T.J. Lee, Y. Pouliot, V. Wagner, P. Gupta, D.W.J. Stringer-Calvert, J.D. Tenenbaum, and P.D. Karp. BioWarehouse: a bioinformatics database warehouse toolkit. *BMC bioinformatics*, 7(1):170–184, 2006.

[LSR03] L. Li, C.J. Stoeckert, and D.S. Roos. OrthoMCL: identification of ortholog groups for eukaryotic genomes. *Genome research*, 13(9):2178–2189, 2003.

[MCP06] Á. Moraga, C. Calero, and M. Piattini. Comparing different quality models for portals. *Online Information Review*, 30(5):555–568, 2006.

[Mou04] D.W. Mount. *Bioinformatics: sequence and genome analysis.* CSHL press, 2004.

[MSRO+10] P. Missier, S. Soiland-Reyes, S. Owen, W. Tan, A. Nenadic, I. Dunlop, A. Williams, T. Oinn, and C. Goble. Taverna, reloaded. In *Scientific and Statistical Database Management*, pages 471–481. Springer, 2010.

[MWP+10] C. Murphy, M. Wu, J. Powlowski, G. Butler, and A. Tsang. Curation of Fungal Glycoside Hydrolase Genes and the Development of mycoCLAP, a Searchable Database for Characterized Lignocellulose-Active Proteins. `http://www.cellulosic-biofuel.ca/ocs/index.php/agm/2010/paper/view/122`, March 2010. Online; accessed 19-July-2010.

[NMM+09] B. Neron, H. Menager, C. Maufrais, N. Joly, J. Maupetit, S. Letort, S. Carrere, P. Tuffery, and C. Letondal. Mobyle: a new full web bioinformatics framework. *Bioinformatics*, 25(22):3005–3011, 2009.

[Ola05] P.I. Olason. Integrating protein annotation resources through the Distributed Annotation System. *Nucleic acids research*, 33(Web Server Issue):W468–W470, 2005.

[PDK+07] A. Prlić, T.A. Down, E. Kulesha, R.D. Finn, A. Kähäri, and T.J.P. Hubbard. Integrating sequence and structural biology with DAS. *BMC bioinformatics*, 8(1):333, 2007.

[PR04] G.A. Petsko and D. Ringe. *Protein structure and function.* Sinauer Associates Inc, 2004.

[Prl07] A. Prlic. Dasobert DAS client library. `http://www.spice-3d.org/dasobert/`, 2007. [Online; accessed 15-June -2010].

[RLB$^+$00] P. Rice, I. Longden, A. Bleasby, et al. EMBOSS: the European molecular biology open software suite. *Trends in genetics*, 16(6):276–277, 2000.

[RLG$^+$06] M. Reich, T. Liefeld, J. Gould, J. Lerner, P. Tamayo, and J.P. Mesirov. GenePattern 2.0. *Nature genetics*, 38(5):500–501, 2006.

[SEOK96] G.D. Schuler, J.A. Epstein, H. Ohkawa, and J.A. Kans. Entrez: molecular biology database and retrieval system. *Methods in Enzymology*, 266:141–162, 1996.

[Sko06] A. Skonnard. Service Station-Web Service Software Factory. *MSDN Magazine-Louisville*, pages 125–132, 2006.

[SLDL06] S. Stephens, D. LaVigna, M. DiLascio, and J. Luciano. Aggregation of bioinformatics data using Semantic Web technology. *Web Semantics: Science, Services and Agents on the World Wide Web*, 4(3):216–221, 2006.

[SN87] N. Saitou and M. Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular biology and evolution*, 4(4):406–425, 1987.

[SRO03] M. Senger, P. Rice, and T. Oinn. Soaplab-a unified Sesame door to analysis tools. In *Proceedings of the UK e-Science All Hands Meeting, Nottingham, UK 2-4 Sept*, 2003.

[Sub98] S. Subramaniam. The Biology Workbencha seamless database and analysis environment for the biologist. *Proteins: Structure, Function, and Bioinformatics*, 32(1):1–2, 1998.

[TMMF09] W. Tan, P. Missier, R. Madduri, and I. Foster. Building scientific workflow with Taverna and BPEL: A comparative study in caGRID. In *Service-Oriented Computing–ICSOC 2008 Workshops*, pages 118–129. Springer, 2009.

[Tsa09] A. Tsang. Genomic and proteomic approaches to explore the diversity of cellwall-degrading enzymes. `http://dev.gqinnovationcenter.com/documents/rendezVous/proteomics2009Tsang.pdf`, 2009. [Online; accessed 10-June-2010].

[VdV04] E. Van der Vlist. *RELAX NG*. O'Reilly Media, 2004.

[WGFS03] M.D. Wilkinson, D. Gessler, A. Farmer, and L. Stein. The BioMOBY project explores open-source, simple, extensible protocols for enabling biological database interoperability. In *Proc Virt Conf Genom and Bioinf*, volume 3, pages 16–26, 2003.

[WL02] M.D. Wilkinson and M. Links. BioMOBY: an open source biological web services proposal. *Briefings in bioinformatics*, 3(4):331–341, 2002.

[WUP09] W. Wu, T. Uram, and M.E. Papka. Web 2.0-based social informatics data grid. In *Proceedings of the 5th Grid Computing Environments Workshop*, pages 1–7. ACM, 2009.

[ZCT09] Z. Zhang, K.H. Cheung, and J.P. Townsend. Bringing Web 2.0 to bioinformatics. *Briefings in bioinformatics*, 10(1):1–10, 2009.

[ZFD+06] J. Zhang, L. Feuk, GE Duggan, R. Khaja, and SW Scherer. Development of bioinformatics resources for display and analysis of copy number and other structural variants in the human genome. *Cytogenetic and genome research*, 115(3-4):205–214, 2006.