AGH University of Science and Technology
in Krakow

Faculty of Electrical Engineering, Automatics, Computer Science and Electronics

Institute of Computer Science

Tomasz Jadczyk

Bioinformatics Applications in the Virtual Laboratory

MSc Thesis

Major: Computer Science
Specialization: Distributed Systems and Computer Networks

Supervisor:
dr. Marian Bubak

Consultancy:
dr. Maciej Malawski

Album id: 127116

Krakow 2009

Oświadczenie autora

Oświadczam, świadomy odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonałem osobiście i samodzielnie i że nie korzystałem ze źródeł innych niż wymienione w pracy.

Tomasz Jadczyk

Akademia Górniczo - Hutnicza
im. Stanisława Staszica
w Krakowie

Wydział Elektrotechniki, Automatyki, Informatyki i Elektroniki

Katedra Informatyki

Tomasz Jadczyk

Aplikacje bioinformatyczne w wirtualnym laboratorium

Praca magisterska

Kierunek: Informatyka
Specjalność: Systemy rozproszone i sieci komputerowe

Promotor:
dr inż. Marian Bubak

Konsultacja:
dr inż. Maciej Malawski

Nr albumu: 127116

Kraków 2009

Abstract

Bioinformatics is the field of science in which biology, computer science, and information technology merge to form a single discipline. Its main focus is on broadening the spectrum of biological analysis by facilitating the new discoveries within its field, as well as providing a more universal perspective of research which would spur the process of revealing the unifying principles in biology. The virtual laboratory is an environment where the applications are available as reusable components (gems) and are exploited within *(in silico)* experiments.

The subject of this thesis is to prepare a set of applications suited to solve common problems in the field of bioinformatics, as well as to integrate them into the virtual laboratory. Three main problems tackled in the scope of this thesis: protein sequence and structure comparison, ligand binding site prediction and microarray analysis, have led to the selection of the appropriate applications. The applications responsible for data gathering, preliminary data analysis and visualization of the results are also included in the prepared set. In addition, the work is focused on integrating selected applications into the virtual laboratory with the use of the available technologies, as well as preparing additional mechanisms that would ensure the proper working of the applications. In the course of this thesis the created experiments that cover the problems listed above are also presented. The classification of the applications is proposed in order to select the most appropriate set of applications that can be used in a variety of bioinformatics experiments.

Contents of this thesis is organized as follows: Firstly, the background of the target environment and elementary bioinformatics knowledge are introduced. Then we are proceeding to the analysis of the related work and the examination of the problem, which leads to providing the classification of applications by technology and by the scope of use. A description of the additional mechanisms required to integrate the selected applications is followed by the presentation of database access and basic analysis gems. After that, the solutions of main bioinformatics problems discussed in this thesis are offered. Finally, the applications for data visualization are listed.

Keywords: bioinformatics, virtual laboratory, ViroLab, experiment, sequence alignment, structure alignment, binding site prediction, microarray

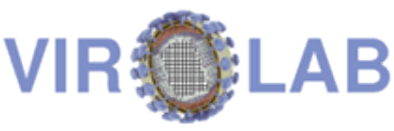
Acknowledgements

I would like to kindly acknowledge my grateful thanks to dr. Marian Bubak, the supervisor of this work, for his guidance, advices and time. Especially, I would like to express my gratitude to dr. Maciej Malawski for the invaluable commitment, insightful look and very helpful collaboration.

The author wishes to appreciate contributions from the Department of Bioinformatics and Telemedicine, Jagiellonian University – Medical College, especially prof. Irena Roterman-Konieczna, dr. Monika Piwowar and Katarzyna Prymula because without them, this work would not be possible.

Also, the help of all the ViroLab Virtual Laboratory team members from ACC Cyfronet AGH cannot be overestimated.

This work was made possible owing to the ViroLab (`http://www.virolab.org`) and PL-Grid (`http://www.plgrid.pl/`) projects.



Contents

Abstract . . . 5
Acknowledgements . . . 6
List of Figures . . . 9
List of Tables . . . 11
Abbreviations and Acronyms . . . 12
Chapter 1. Introduction . . . 14
1.1. Science of Bioinformatics . . . 14
1.2. ViroLab - Target Environment . . . 15
1.3. Problem Description . . . 17
1.4. The MSc Thesis Objectives . . . 18
1.5. Organization of the Thesis . . . 18
Chapter 2. Background of Bioinformatics and ViroLab Virtual Laboratory . . . 19
2.1. Bioinformatics – Theoretical Background . . . 19
2.2. ViroLab Virtual Laboratory . . . 27
2.3. Summary . . . 30
Chapter 3. Related work . . . 31
3.1. Soaplab - SOAP-based Analysis Web Service . . . 31
3.2. Taverna . . . 32
3.3. Biomedical Informatics Research Network . . . 34
3.4. META-PredictProtein . . . 36
3.5. Summary . . . 36
Chapter 4. Analysis of Bioinformatics Applications and Gems . . . 37
4.1. Model of Bioinformatics Applications . . . 37
4.2. Classification of Gems . . . 40
4.3. Additional Integration Mechanisms . . . 47
4.4. Summary . . . 51
Chapter 5. Database Access Layer and Basic Analysis Layer Gems . . . 52
5.1. Bioinformatics Database Access Gems . . . 52
5.2. Statistical Analysis . . . 59

5.3. Data Mining . . . 62
5.4. Data Clustering . . . 67
5.5. Reduction of Data Dimension . . . 68
5.6. Summary . . . 68
Chapter 6. Protein Sequence and Structure Comparison . . . 69
6.1. Problem Description . . . 69
6.2. Experiment . . . 73
6.3. Gems and Classes Used in Experiment . . . 78
6.4. Experiment Run and Results . . . 85
6.5. Summary . . . 88
Chapter 7. Comparison of Services for Predicting Ligand Binding Site . . . 89
7.1. Problem Description . . . 89
7.2. Description of Available Services . . . 90
7.3. Integration of Gems Using Task Queuing System . . . 94
7.4. The Prediction Experiment of Ligand Binding Sites . . . 100
7.5. Summary and Results . . . 103
Chapter 8. Microarray Data Analysis . . . 105
8.1. Problem Description . . . 105
8.2. Microarray Analysis in Virtual Laboratory . . . 106
8.3. Sample Microarray Experiment in Virtual Laboratory . . . 113
8.4. Summary . . . 113
Chapter 9. Results Presentation Layer . . . 115
9.1. Plotting Numerical Data With Gnuplot . . . 115
9.2. Protein Structure Visualization with Jmol and ProteinWorkshop . . . 116
9.3. Sequences Alignment in Jalview . . . 120
9.4. Microarray Clustering Results in JTreeView . . . 121
9.5. Summary . . . 121
Chapter 10. Conclusions and Future Work . . . 123
10.1. Summary . . . 123
10.2. Future work . . . 125
Bibliography . . . 127
Appendix A. Glossary . . . 131
Appendix B. The experiments . . . 136
B.1. ProteinStructureAndSequenceComparison . . . 136
B.2. ProteinPocketsTests . . . 147
Appendix C. Gems API . . . 158
C.1. Bioinformatic database access gems . . . 158
C.2. Basic analysis gems . . . 160
C.3. Protein sequence and structure analysis gems . . . 163
C.4. Protein binding site prediction gems . . . 166
C.5. Results presentation gems . . . 169

List of Figures

1.1. ViroLab virtual laboratory layered architecture 16

2.1. The Central Dogma of Molecular Biology 21
2.2. Transfer RNA (tRNA) molecule 22
2.3. Protein synthesis process 22
2.4. Four levels of protein structure description 24
2.5. DNA replication process 25
2.6. Mutations in amino acid sequence for hemoglobin protein 26
2.7. Grid Object abstractions 28
2.8. Experiment execution modes 29

3.1. Taverna workbench 34

4.1. General model of bioinformatics experiment process 39
4.2. Layers of the gem technology 43
4.3. Layers of the gem scope of usage 45
4.4. Bioinformatics database access layer 46
4.5. Basic analysis layer 46
4.6. General architecture of the tasks queueing system 48

5.1. ScopDb query execution diagram 59
5.2. R gem running diagram 61
5.3. Class diagram of Weka data handling part 63
5.4. Weka Mocca components. 64
5.5. The experiment with comparison of Weka Classifiers 66

6.1. Two paths of protein structure analysis 71
6.2. Scenario of the Protein Sequence and Structure Comparison experiment 74
6.3. FASTA format example 75
6.4. Example of sequences alignment for protein family 76
6.5. W score computing method 77
6.6. Wprotein profile computing example 77
6.7. Mammoth gem running diagram 82
6.8. MultiProt gem running diagram 83
6.9. Experiment results. W score values 86

6.10. Experiment results. W profiles for 2DD8 protein 87

7.1. Class diagram for binding site prediction services 95
7.2. Sequence diagram for Task creation process 96
7.3. Sequence diagram for running Task analysis 97
7.4. Class diagram for analyzers part of the system. 98
7.5. Class diagram for Options mechanism. 101
7.6. Binding site prediction - Experiment model 101
7.7. Binding site prediction - Results conversion and visualization 103
7.8. Visualization of predicted binding sites for 1ITQ protein. 104

8.1. Microarray data model class diagram 107
8.2. Sequence diagram for creating a new microarray dataset 109
8.3. Class diagram for microarray data clustering library. 111
8.4. The structure of microarray data analysis experiment 113

9.1. An example of the gnuplot script and plot 117
9.2. The Protein Workshop visualization example. 119
9.3. Enhanced Jmol viewer main window. 120
9.4. Gene expression visualization in JTreeView software. 122

List of Tables

2.1. Genetic code, amino acids and assignments of all 64 triplets 20
2.2. Grid Object Implementation - available technologies. 28
2.3. Experiment running modes comparison. 30

4.1. The parameters of a new gem registration 42
4.2. Required parameters during registration process for different gem technologies. 42
4.3. Tasks queuing system classes. 49

5.1. Atom record structure in PDB file. 55
5.2. The complete list of accepted FASTA codes. 56
5.3. Conversion between data formats rules 60

7.1. Binding site prediction services - format description and rules of conversion . . 93
7.2. Parameters values for available binding site prediction services. 100

10.1. Created gems statistics, classification by technology 124
10.2. Created gems statistics, classification by scope of usage 125

Abbreviations and Acronyms

Acronym	Meaning
AA	Amino acid
API	Application Programming Interface
ARFF	Attribute-Relation File Format for Weka library
BLOSUM	Blocks of Amino Acid Substitution Matrix
CCA	Common Component Architecture
cDNA	complementary DNA
CORBA	Common Object Request Broker Architecture
CSV	Comma Separated Values
DNA	Deoxyribonucleic acid
EBI	European Bioinformatics Institute
EMI	Experiment Management Interface
EPE	Experiment Planning Environment
FIFO	First-In-First-Out
FTP	File Transfer Protocol
GEO	Gene Expression Omnibus
GRR	Grid Resource Registry
HIV	Human Immunodeficiency Virus
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IDL	Interface Definition Language
JAR	Java Archive
JDBC	Java Database Connectivity
JNLP	Java Network Launching Protocol
MDS	Multidimensional scaling
MIME type	An Internet media type
MOCCA	Metacomputing-oriented CCA framework
mRNA	messenger RNA
MSA	Multiple Sequence Alignment

Acronym	Meaning
NCBI	The National Center for Biotechnology Information
NP	Non-deterministic Polynomial-time
PAM	Point Accepted Mutation
PCA	Principal component analysis
PDB	Protein Data Bank
RCSB	Research Collaboratory for Structural Bioinformatics
REST	Representational State Transfer
RMSD	Root mean square deviation
RNA	Ribonucleic acid
RPC	Remote Procedure Call
SOAP	originally Simple Object Access Protocol, lately also Service-Oriented Architecture Protocol
SQL	Structured Query Language
SSM	Secondary Structure Matching
tRNA	transfer RNA
UDDI	Universal Description Discovery and Integration
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VLvl	ViroLab Virtual Laboratory
WebDav	Web-based Distributed Authoring and Versioning
WSDL	Web Services Description Language
WWW	World Wide Web
XML	Extensible Markup Language

Chapter 1

Introduction

The subject of this thesis is to prepare a set of applications suited to solve common problems in the field of bioinformatics, as well as to integrate them into the virtual laboratory. This Chapter presents general information about bioinformatics as a single discipline that emerged from biology, computer science and information technology. Next, the environment where the applications will be running - ViroLab Virtual Laboratory, its architecture and users, is introduced. Finally, the problem of this thesis is stated and that goals that are to be achieved are listed.

1.1. Science of Bioinformatics

Bioinformatics is a relatively new field of science that has rapidly advanced in recent years. Therefore it is quite difficult to give its complete definition. In this section two definitions of bioinformatics and its exploration areas are presented.

The first stage of bioinformatics evolution was focused on providing access to data for researchers [15]. Within this scope the following definition is commonly accepted:

Bioinformatics *is a science focused on creating biological databases and web-based front ends available to any user interested in searching and modifying biological data*

When the data availability has reached a satisfying level, the next stage of development in the field of bioinformatics approached. It was focused on providing and developing methods and algorithms purposed to data analysis. The most appropriate definition, describing bioinformatics at this stage of evolution is presented in [35]:

Bioinformatics *is an interdisciplinary science, that covers two issues:*

- *development of computing methods to analyse structure, function and evolution of genes, proteins and genomes,*
- *development of methods used in management and analysis of biological information data which is gained in the process of genomics studies and research conducted with the use of high-throughput experimental techniques.*

Biology is a primary science from which bioinformatics has evolved, but more particular biology domain on which bioinformatics is based are population genetics and molecular evolution.

It can be observed that the amount of data available in public and specialised databases is rising exponentially. Moreover, many new algorithms and applications are also being developed. We can relate dynamic development of bioinformatics with simultaneous evolution of the Internet. Thanks to the global network, the biological databases with DNA sequential data has been available to all interested users all over the world.

Among the main bioinformatics research areas we can list:

- Putting gene and protein data into public databases, providing interfaces for searching, adding and revising those data,
- Sequence alignment,
- Searching similar sequences,
- Phylogenetic analysis,
- Protein structure and function analysis,
- Protein structure prediction,
- Protein-protein interaction network modelling,
- Gene and protein expression analysis,
- Computer-aided drug research.

More detailed description of bioinformatics backgrounds is presented in section 2.1.

1.2. ViroLab - Target Environment

The ViroLab Virtual Laboratory, which is the target environment for bioinformatics applications in the scope of this thesis, is a part of ViroLab project [7]. This section[1] intends to demonstrate its architecture and users.

ViroLab virtual laboratory is a set of integrated components that form an advanced environment, which can be used to plan and perform complex scientific applications, called **experiments**. Experiments in this context are processes that combine available applications, services, tools and data into ordered activities sequence and perform computer simulation (*in silico*) in order to obtain new knowledge.

A mission of the ViroLab project is to provide support for virologists, epidemiologists and clinicians investigating the HIV virus and the possibilities of treating HIV-positive patients. The offered solution is general enough to reuse it in other domains among which bioinformatics is perhaps the best example.

[1] This section is based on [7, 50].

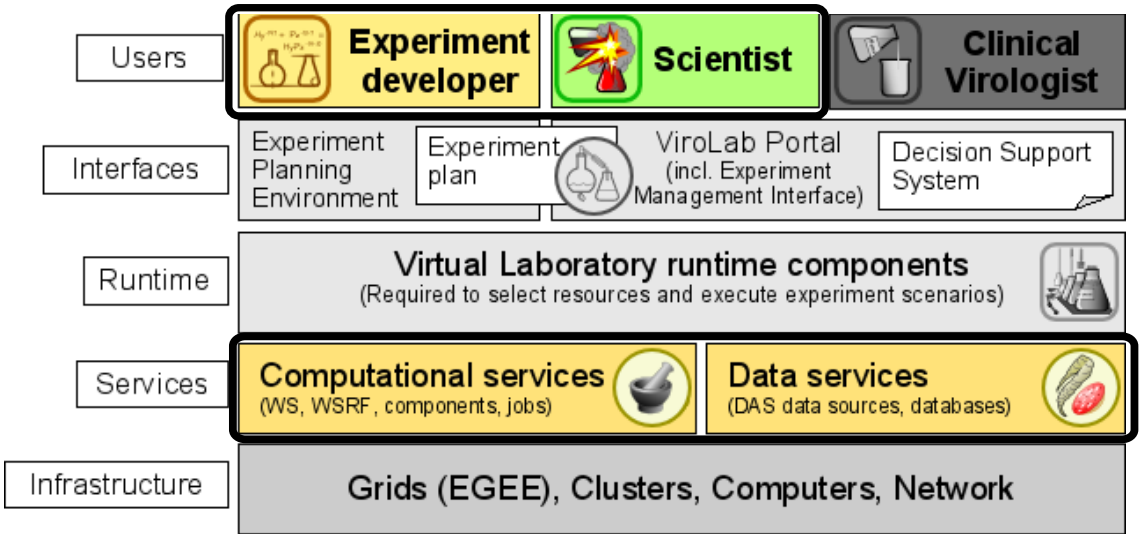


Figure 1.1. ViroLab virtual laboratory layered architecture. Emphasized layers are the main areas in the scope of this thesis. Picture comes from VLvl homepage [12].

The ViroLab virtual laboratory architecture is presented in Fig. 1.1. It is a conceptual way to show abstract layers of the virtual laboratory, however the real architecture is far more complex.

The areas of research which are of particular concern in this thesis are included in the highlighted layers. Computational services, called **gems**, allow **experiment developers** to plan and carry out experiments. The experiments need input data to work on. Those data can be gathered from several databases by using gems from *Data services* layer. Prepared and published experiments are performed by **scientists**. They have to know which pieces of data should be analysed and they are able to explain the results of the experiment.

There are two main classes of the virtual laboratory users:

- *Experiment developer* - a person who designs experiments in a specific domain. This person must be skillful enough to design and denote the way in which an experiment should proceed, have a certain level of domain-related knowledge to understand the nature of the processes in order to model an experiment and technical skills to use available services in the virtual laboratory. The experiments are written in Ruby language [10] and can be created in ViroLabEPE environment [11]. In the scope of this thesis, its author is taking an *experiment developer* role.
- *Experiment user* - is any person who runs a previously prepared and published experiment in order to obtain results. In the scope of this thesis *experiment users* are the scientists from Department of Bioinformatics and Telemedicine of Jagiellonian University - Medical College [3], under professor Irena Roterman – Konieczna guidance, who have run the experiments prepared and suited to their research areas, repeating the process many times on different input data sets.

Virtual laboratory experiments are developed in ViroLabEPE environment and can be run in this environment as well as by using Experiment Management Interface (EMI) portal. More detailed description of the ViroLab virtual laboratory is presented in section 2.2.

1.3. Problem Description

Section 1.1 of this Chapter familiarized us with bioinformatics as a multi-faceted science. What is more, its dynamic development is also a key feature which stimulates creation of new applications aimed at finding new solutions to bioinformatics problems. One subset of these applications is perceived by scientists as a set of basic tools that have gained common acceptance in the field. The remaining part is under constant development and testing. A multitude of available applications solving the same problem makes it possible to perform a comparative analysis of many methods and algorithms. However, usually it is necessary to do some additional work in order to compare results. Modern bioinformatics struggles with the following typical problems: an integration of many applications to solve a single issue, a need for using many databases in order to gather data and check results, and finally an issue of providing and maintaining bioinformatics software.

First of all, an integration of many bioinformatics applications may be a problem quite difficult to cope with. The available applications are made using many technologies. Some of them are developed as binary programs, others are provided as compiled libraries which we can use in our programs. The programs are also often delivered as Web services or WWW portals. Every single problem in the field of bioinformatics frequently requires the use of many applications to solve it. The second issue in this area is passing data between programs. Such type of data processing, called pipelining, requires adjusted format of inputs and outputs of the used programs. A communication between them is often achieved by passing data by the local file system.

The second problem is in the necessity of obtaining large amounts of data from many sources to perform an experiment. What is more, a communication established between many databases to gain information needed during the experiment, or comparison of the obtained results with the already existing data, may be done in a variety of ways. The authors of bioinformatics software often prepare Web services in SOAP or REST technology to use their programs. In some databases their own searching systems are available (e.g. Entrez [62] system in NCBI portal) and therefore wrapping HTTP communication is often required to reach it.

Finally, in order to use it, all the needed software should be available for the researcher on a workstation where the work is carried out. Giving access to the most important applications in a unified way would allow to use it many times in different experiments without the need for introducing additional changes to the experiment code. As a result, there would be only one environment necessary to run any experiment, which would be much easier to maintain and learn by new users.

All these problems make the development of bioinformatics applications a challenge, which can be addressed by using a generic framework and by providing mechanisms recognized as objectives of this thesis.

1.4. The MSc Thesis Objectives

The main goal of the thesis is to provide a set of applications which can be used in bioinformatics experiments within the virtual laboratory. In order to define properly this set of applications and to integrate it into the virtual laboratory some sub-goals must be achieved. The goals are as follows:

- Analysis of the available bioinformatics applications, the way in which each application should be run, the available options to set, and the way of returning the results and the results' format;
- Classification of the bioinformatics applications and their division into a number of categories, while taking into consideration the technology in which the application is created, the usage scope and their usefulness in different problems;
- Integration of selected applications projects. Preparing additional mechanisms to run the selected applications as gems in the virtual laboratory;
- Creating a set of ViroLab gems and preparing experiments. Wrapping the selected applications and publishing them into the ViroLab virtual laboratory. Creating the experiments to show how the prepared gems could be used and planning experiments to solve the selected bioinformatics problems;
- Preparing general methods and tools to make using the bioinformatics applications easier in the virtual laboratory experiments.

1.5. Organization of the Thesis

The contents of this thesis is organized in a following way. In Chapter 1 bioinformatics domain and target environment are introduced. Then goals of this thesis are stated. Chapter 2 presents the background, that is elementary bioinformatics knowledge and the ViroLab virtual laboratory architecture. Other bioinformatics frameworks are overviewed in Chapter 3. Chapter 4 focuses on the analysis of the solution and its design. In Chapter 5 the basic gems are described: Database access (section 5.1), Statistical analysis (5.2), Data mining (5.3) Clustering (5.4) and Reducing Dimensionality (5.5) are included in the basic gems set. Further Chapters demonstrate the solutions prepared to solve specialized problems, the available gems, the solutions design, the prepared experiments and the results of the performed experiments. In particular, they include a comparison of proteins on three levels of protein description in Chapter 6, a test of different methods of ligand binding site prediction in proteins in Chapter 7 and a demonstration of an analysis of a large amount of data from microarray experiments in Chapter 8. One of the main part of bioinformatics experiments - results visualization and a way of integrating the visualization tools into the virtual laboratory - are presented in Chapter 9. Finally, Chapter 10 provides concluding remarks and outlines some further issues.

Chapter 2

Background of Bioinformatics and ViroLab Virtual Laboratory

This Chapter introduces the basic terms used in bioinformatics. It explains how biological data is organised in living cells, characterizes The Central Dogma of Molecular Biology, the protein structure and the process of translating those data into the format which can be stored and analyzed in (in silico) experiments. Secondly, a detailed description of the ViroLab virtual laboratory, introduced in the section 1.2, is given. It contains instructions how to extend virtual laboratory capabilities by adding new applications as ViroLab gems, how to use them in the experiments and the tools available to facilitate preparation and execution of an experiment.

2.1. Bioinformatics – Theoretical Background

Using computer techniques to analyze biological data was enforced by rapid increase of the available data to study. Moreover, new knowledge gained by researchers also stimulates development of new methods for data analysis. This section[1] provides an outline of basic information required to understand bioinformatics research area.

Information management in living cells. Living organisms have an ability to store, utilize and pass on information. Information is stored in **genes** and maintained in genetic material - **DNA**. That information allows organisms to construct living cells and smaller organisms from inanimate molecules. Newly created organisms are able to regulate their internal chemical composition, growth and reproduction. DNA is built from repertoire of four different basic structures, called

[1] Information in this section comes from [14, 35, 41].

First base	Second base				Third base
	U	C	A	G	
U	Phe(F)	Ser(S)	Tyr(Y)	Cys(C)	U
	Phe(F)	Ser(S)	Tyr(Y)	Cys(C)	C
	Leu(L)	Ser(S)	STOP	STOP	A
	Leu(L)	Ser(S)	STOP	Trw(W)	G
C	Leu(L)	Pro(P)	His(H)	Arg(R)	U
	Leu(L)	Pro(P)	His(H)	Arg(R)	C
	Leu(L)	Pro(P)	Gln(Q)	Arg(R)	A
	Leu(L)	Pro(P)	Gln(Q)	Arg(R)	G
A	Ile(I)	Thr(T)	Asn(N)	Ser(S)	U
	Ile(I)	Thr(T)	Asn(N)	Ser(S)	C
	Ile(I)	Thr(T)	Lys(K)	Arg(R)	A
	Met(M)	Thr(T)	Lys(K)	Arg(R)	G
G	Val(V)	Ala(A)	Asp(K)	Gly(G)	U
	Val(V)	Ala(A)	Asp(K)	Gly(G)	C
	Val(V)	Ala(A)	Glu(E)	Gly(G)	A
	Val(V)	Ala(A)	Glu(E)	Gly(G)	G

Table 2.1. Genetic code, amino acids and assignments of all 64 triplets

nucleotides: guanine (**G**), adenine (**A**), thymine (**T**) and cytosine (**C**). During DNA to RNA translation process, presented in next paragraph, thymine unit is replaced by uracil (**U**). Information stored in genes conveys as specific combination of nucleotides - it is simply an order in which those nucleotides are found along DNA molecules. Complicated genes can be many thousands of nucleotides long, and all of the organism's genetic instructions (**genome**) can be maintained in millions or even billions of nucleotides. DNA structure forms a two-strand helix, which is kept thanks to interaction between base pairs of nucleotides - adenine in one strand is connected to thymine in the other, and guanine is connected to cytosine. Two strands are antiparralel and exactly complementary, so the information in DNA is redundant.

As it was mentioned, DNA or RNA are built from nucleotides. On the contrary, proteins, other very important group of macromolecules, are constructed from the set of 20 different units, called **amino acids**. Successive nucleotides triplets in RNA sequences form amino acids coding units - **codons**. Almost every amino acid can be built from more than one coding sequence; there are also functional sequences: one to mark where a gene starts (AUG) and three codons to mark the end of coding sequence (UAA, UAG, UGA). The genetic code is nearly universal for all the organisms. Only few of them have formed small modifications for limited number of amino acids to codons assignments. Not the whole of DNA sequence contains usable information. A large part of DNA are non-coding sections, called introns. These regions are subsequently removed during the transcription process. Only coding parts - exons - are left in a final mRNA sequence.

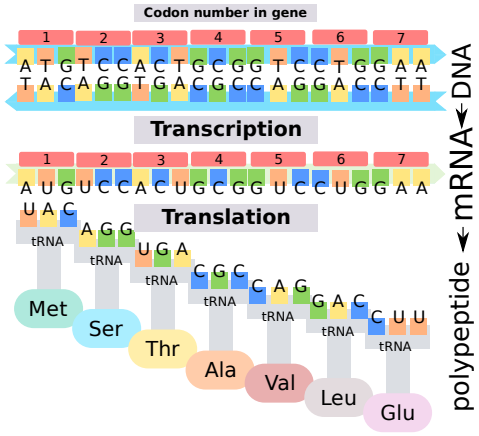


Figure 2.1. The Central Dogma of Molecular Biology. Information in cells passes from DNA to RNA to proteins. RNA is made from DNA molecules during transcription by RNA polymerases. Proteins are made from the information content of RNA molecules as they are translated by ribosomes.

The Central Dogma of Molecular Biology. There is a statement called *Central Dogma of Molecular Biology*, which depicts how genetic information in cells is passed from DNA to RNA and then to proteins. The process of making an RNA copy of a gene is called **transcription**. The process of converting that information from nucleotide sequences in RNA to the amino acid sequences that make a protein is called **translation**. RNA strand transcripted from DNA coding region is called messenger RNA (mRNA). Within eukaryotes DNA is placed in nucleus, where both transcription and processing of RNA takes place. Newly synthesized mRNA molecules are known as primary transcripts or pre-mRNA. They must undergo post-transcriptional modification in the nucleus before being exported to the cytoplasm, where the translation process is performed. The model of this process is presented in Fig. 2.1.

Translation is a protein synthesis process, based on information from mRNA sequence. To perform the translation process transfer RNA (tRNA) molecules are used. Transfer RNA molecule contain two main regions in its structure - a site for amino acid attachment and three base regions called **anticodon**s. Anticodon can bind to the complementary part of codon region on mRNA (basic pairing rules are A-U and C-G). For example UGC anticodon can pair with GCA sequence (anticodon is presented in the opposite direction, real pairing is CGU-GCA). It can be seen from Table 2.1 that codon GCA corresponds to alanine (Ala). Another example is given in Fig. 2.2, where tRNA molecule specific for phenylalanine, with anticodon AAA that pairs with UUU, is shown.

Protein biosynthesis occurs in the cytoplasm, where the ribosomes are located. Ribosome is made of a small and large subunit that surrounds mRNA. It moves along mRNA sequence with one codon step. tRNA with appropriate amino acid is selected and inserted into the ribosome; the selection is based on anticodon region.

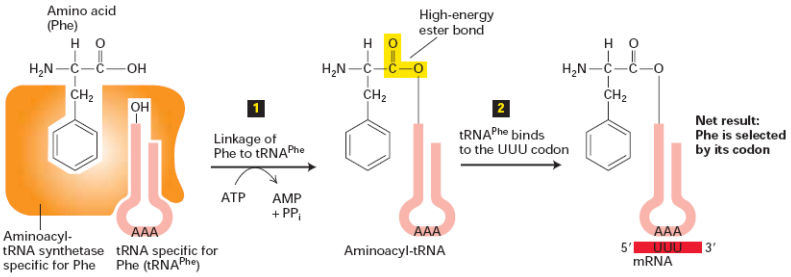


Figure 2.2. Transfer RNA (tRNA) molecule [14]. Anticodon region consists of three adenine nucleotides, phenylalanine molecule is attached to the amino acid side, tRNA binds to the UUU codon.

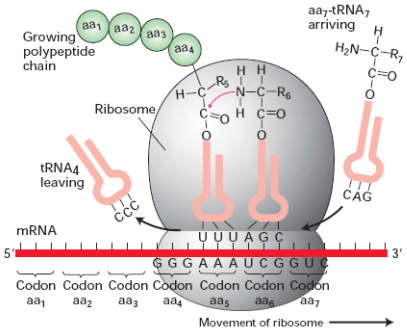


Figure 2.3. Protein synthesis process [14]. Ribosome is made from a small and a large subunit. Molecule moves along mRNA and binds amino acid from appropriate tRNA to the growing protein chain.

The amino acid from the other site is then cut off from the tRNA and bound to polypeptide chain. tRNA is then removed from the ribosome which moves along the mRNA sequence for another nucleotide triplet and next tRNA is bound to the ribosome. When the ribosome meets STOP codon on the mRNA, a special protein, called release factor is bound instead of tRNA and the ribosome is detached from the mRNA strand. Scheme of protein synthesis is shown in Fig. 2.3 . Ribosome molecules are catalysts in polypeptide chain creation process. Ribosomal RNA (rRNA), central component of ribosome is called ribozyme by analogy to protein enzymes.

Protein - functions and structure. Proteins are the molecular machinery responsible for performing most of the cell work. Their functions are incredibly diverse and depend on proteins structure. Structural proteins, such as collagen, provide rigidity and support in bones and connective tissues; enzymes act as biological catalysts in a variety of reactions (e.g. pepsin helps in metabolism process); others are responsible for transportation of atoms and small molecules throughout an organism (e.g. hemoglobin). Some other responsibilities of proteins are signalling and

inter-cellular communication (e.g. insulin), absorbing photons (e.g. rhodopsin), and many other functions.

Proteins are built from 20 different amino acids. Amino acids have similar structure; a central carbon atom, called the α-carbon, has attached an amino group, a carboxylic acid group, a hydrogen atom and a distinctive R group, also referred as **side chain**. Therefore amino acids differ in structure of the side chain, that can vary. A common part that is presented in each amino acid is called **backbone**. The backbone parts can vary in size from just a hydrogen atom in glycine through a methyl group in alanine to a large heterocyclic group in tryptophan. Proteins can be built even from thousands of amino acids, despite that they adopt compact structures. After the translation process, the created polypeptide chain does not remain as plain structure, but folding process has to be performed. Proteins are often complex structures, their shape depends on amino acid sequence and properties. From these properties we can mention amino acid size (structure of protein is quite compact, next amino acid residue is placed in to fill all available space), electronegativity, polarity, and amino acid hydrophobicity, that is their abilities to interact with water molecules. Polar amino acids are hydrophilic ("water friendly") and are often present on protein surface, Non-polar amino acids, hydrophobic, are placed inside of protein.

Protein structure can be described on four different levels:

Primary structure It is an order in which different amino acids are assembled into a protein, another name of this level is **amino acid sequence**

Secondary structure Chemistry of amino acids structure makes that large part of backbone is rigid and only few places (especially bonds to alpha carbon) are mobile. Examination of the proteins whose structures are known reveals that a small number of patterns in local structures are quite common. The most common are the α-helix and the β-sheet.

Tertiary structure The regions of the secondary structure in a protein packed together and combined with other, less structured regions form an overall three–dimensional shape, where each atom is placed in available space.

Quaternary structure A protein is often composed from more than one polypeptide chain. The overall structure formed by the interacting proteins is called quaternary structure.

An example of protein structure description is presented in Fig. 2.4.

DNA replication, mutations and evolutions. Each cell division enforces DNA replication, which is a fundamental process occurring in all living organisms an aiming at copying their DNA. This process is "semiconservative", which means that each strand of the original double-stranded DNA molecule serves as template for the reproduction of the complementary strand. Hence, following DNA replication, two identical DNA molecules have been produced from a single double-stranded DNA molecule. The replication process is initiated at particular points within the DNA, known as "origins", which are targeted by proteins that separate the two strands and initiate DNA synthesis. When DNA is being replicated it forms special

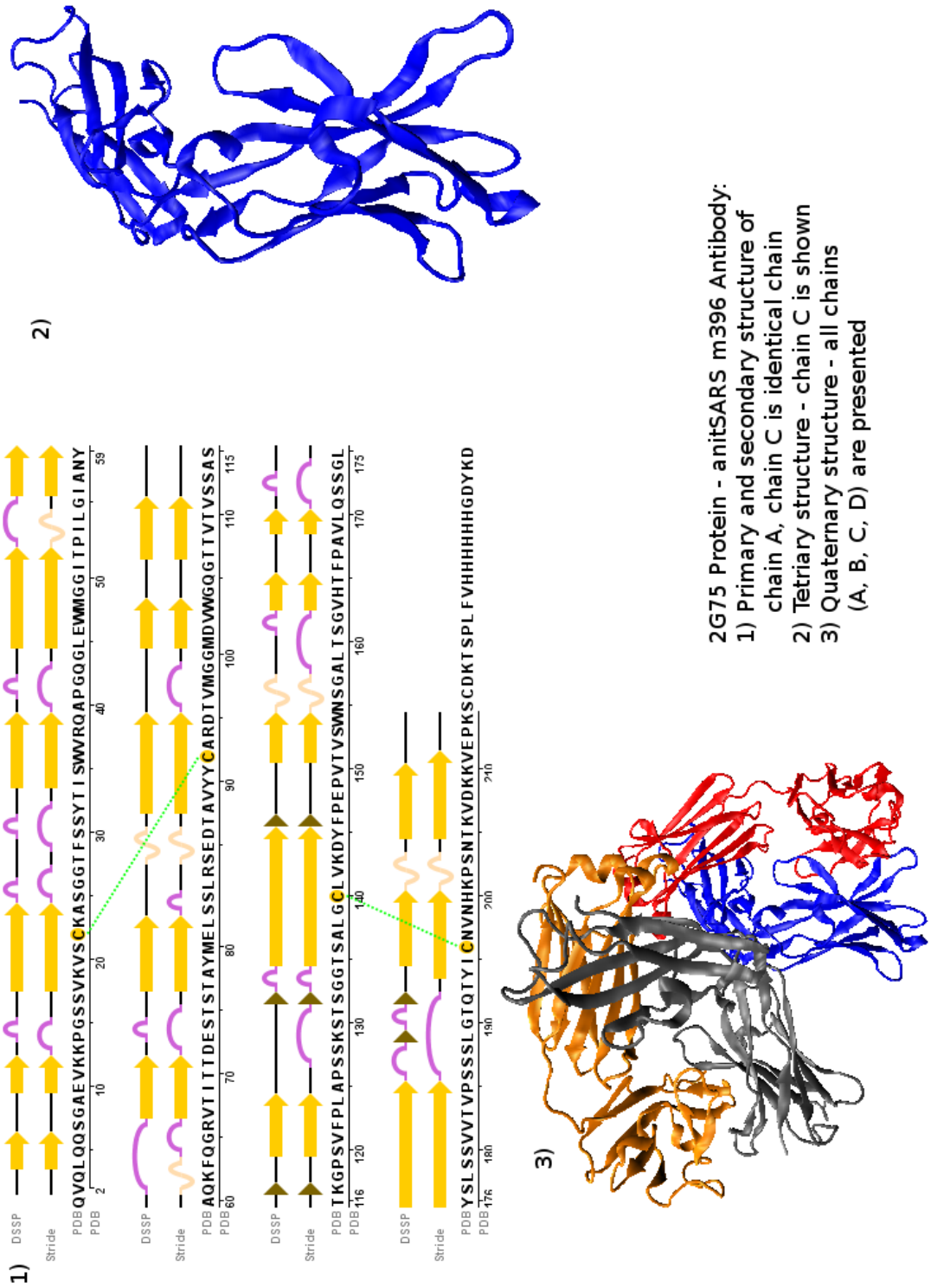


Figure 2.4. Four levels of protein structure description. The primary and secondary structures come from PDB [9], while tetriary and quaternary structures are visualizations in Jmol [5].

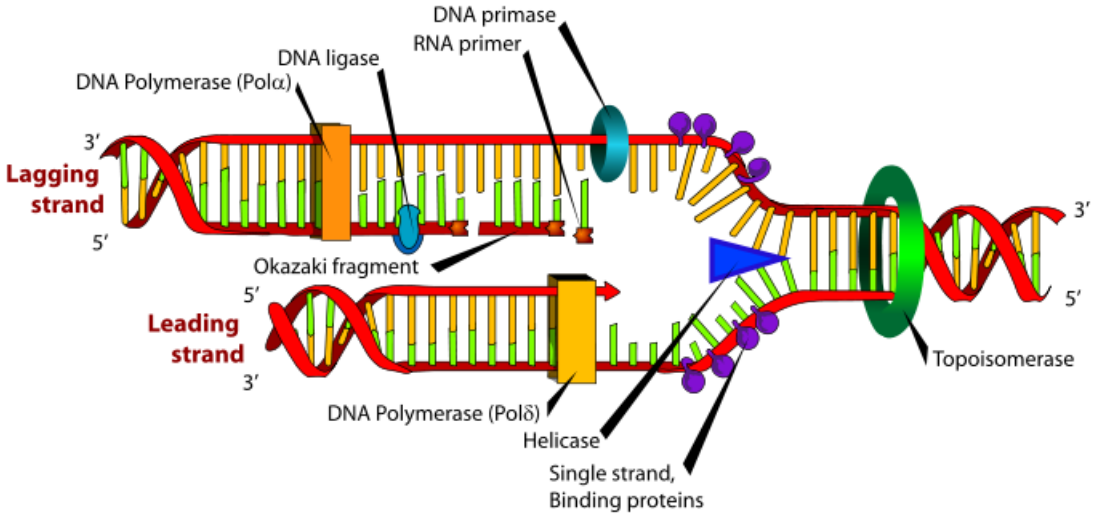


Figure 2.5. DNA replication process. Picture comes from http://en.wikipedia.org/wiki/DNA_replication .

structure called **replication fork**, presented in Fig. 2.5. Primary enzyme complex involved in DNA replication is known as DNA polymerase. A special short sequence, called **primer** is required to start the replication process. During a proof-reading process DNA polymerase is able to remove nucleotides from the end of a newly created DNA strand if they do not match to the original strand. As a result of using this mechanism, DNA replication process is remarkably more accurate than RNA translation. An error in DNA replication causes hereditary point mutation, while RNA translation and synthesis error results only in producing faulty, short-lived mRNA molecules. It is the main reason why DNA replication must be very precise.

In spite of using a proof-reading mechanism, replication is a susceptible to errors. This feature leads to incompatibility of that newly created DNA strands with the original DNA. It is a basic property of the replication process that allows organisms to evolve. If a replication was a faultless process, living organisms could not evolve - all of them would be the same. As we can see, replication errors are necessary in the evolutionary process, but their number must be limited because too many errors make another important process, inheritance, impossible. The second important part of the evolution process is a natural selection, thanks to which "positive" heritable traits become more common in a population over successive generations. Any single change in a nucleotide sequence of the genetic material in an organism, that can be passed to a successive generation is called a **mutation**. Point mutations are present at the whole length of DNA. Mutation effect depends on the position in DNA sequence where it occurs. In many genomes large non-coding regions are present in DNA sequences. Mutation in these places probably will not have any influence on any type of protein and will be **neutrareasonal mutation**. But non-coding regions

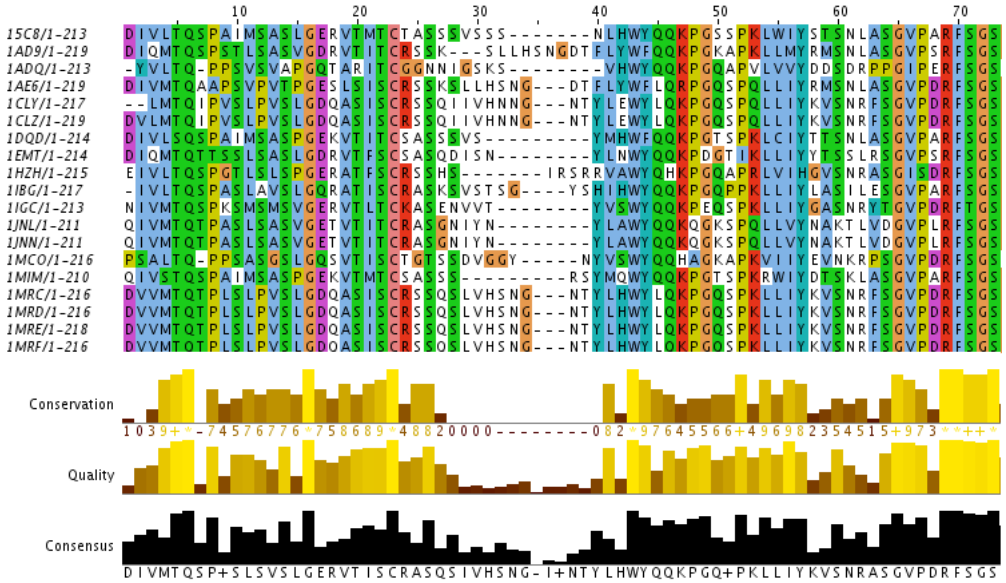


Figure 2.6. Mutations in amino acid sequence within hemoglobin protein family. Selected organisms are: Teleost fish, Emerald rockcod, parrot, rat, and human. Visualization in JalView [67] was created with Jalview gem using (see section 9.3), after sequence alignment performed in ClustalW program [66].

contain also functional signals, that initiate transcription and translation processes. Mutation in these places may have large impact on an organism adaptability. When mutation occurs in the region that codes protein, it may cause change in amino acid sequence of that protein. Genetic code structure allows not every mutation to cause amino acid change; those type of nucleotide exchange are called **silent mutations**. The situation when mutation causes nucleotide exchange is called **missense mutation** - it means that one amino acid is changed to another. This type of mutation can be **harmful** (decreasing organism's fitness) or **beneficial** (due to an increase in organism's fitness and adaptability). In **nonsense mutations** stop codon appears in the middle of gene coding sequence and it is almost always harmful by truncating the protein. The other type of mutations present in replication process are insertions and deletions of nucleotide sequence fragments, called **indels**. One of the reasons why this mutation type happens is polymerase sliding. This type of errors can disrupt reading frame or codons grouping, resulting in a completely different translation from the original, if a number of nucleotides inserted or deleted is not evenly divisible by three, due to the triplet nature of a gene expression by codons. The earlier in the sequence the deletion or insertion occurs, the more altered the protein is produced. The majority of mutations that occur in a gene within the same species is harmful, and most of mutations between species are rather neutral or beneficial because the natural selection does not discard those mutations. Some types of mutations are presented in Fig. 2.6.

2.2. ViroLab Virtual Laboratory

As it was stated in the section 1.2, the ViroLab virtual laboratory is a set of integrated components that, used together, form a distributed and collaborative space for science [33, 23]. Thanks to those components the users are able to run *(in-silico)* experiments. The purpose of the virtual laboratory is to support collaborative work of all the people who are effectively involved in any stage of the experimental process. This section provides a detailed description of ViroLab virtual laboratory components and gives information about experiment running, which are the main entities available in the virtual laboratory.

Experiment is a process that combines together data with a set of activities that act on that data in order to yield experiment results. The substrate data required for an experiment may be obtained from multiple resources in various possible forms. No definite restrictions are imposed on the level of complexity of such an experiment.

Experiments in the ViroLab virtual laboratory are defined by name and experiment version. The main part of an experiment is a **script** file. The script is a program, created in JRuby language, in which virtual laboratory components, called Grid Objects, are used. Scripts can act on data locally available or can get access to data using specialized database access services.

Grid Object is by definition any entity that is accessed remotely from the experiment execution environment that provides computation abilities for the experiment (to distinguish it from data sources). *Grid Object* term is an abstraction; three abstraction levels can be distinguished:

- conceptual level - *Grid Object Class* - it defines an interface which every *Grid Object* that belongs to a certain class exposes. Within its interface every class defines a set of one or more *Grid Operations*. Each operation describes what type of input parameters it requires and what kind of output results it yields upon successful completion. Such a description of functionality is frequently called an *operation signature*;
- realization level - *Grid Object Implementation* - this abstraction may be done in one of supported technologies, presented in Table 2.2. The same *Grid Object Class* may be realized in different technologies, but every implementation provides exactly the same functional behaviour to the external world (it realizes *Grid Object Class* interface);
- execution level - *Grid Object Instance* - is an implementation that was executed and published so that the *Grid Object* can be accessible from the network.

All of those Grid Object abstraction levels are presented in Fig. 2.7. In order to add a new Grid Object the developer needs to provide all of this abstraction levels. The available Grid Objects are registered in Grid Resource Registry (**GRR**), which contains information about conceptual level - classes' interfaces, which are the most important source of knowledge for experiment developer and also give information about realization and execution levels, that are necessary to run an experiment.

Technology	Overall characteristics	Programming platform
Web services	Useful especially for stateless services of relatively fine-grained communication delays (at most 3-4 minutes to response); mainly used in blocking, synchronous mode.	Various languages and command line
MOCCA	Better suited for heavier, more time consuming computations; internal state could be maintained; dynamic deployment possible.	Java and command line
WSRF	Web services resource framework - stateful Web service implementation, useful for long living application - prototype implementation	Globus 4.0, WebSphere Application Server version 6.1, Muse 2.0, WSRF::Lite, WSRF.NET
WTS	Web service integrating with job execution mechanism - useful for long living application that communicate through input and output files	KUL implementation
gLite	useful for long running jobs - prototype implementation	EGEE
AHE	useful for long running jobs and simulations	Clusters, supercomputers, TeraGrid, DEISA

Table 2.2. Grid Object Implementation - available technologies.

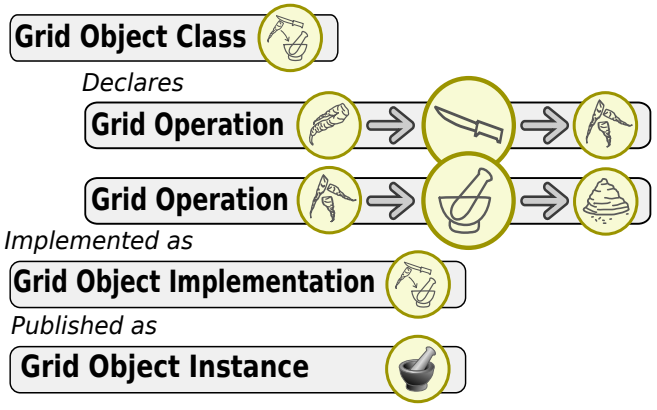


Figure 2.7. Grid Object abstraction, from the VLvl homepage [12]. Grid Object interface is defined as a set of operations within Grid Object Class, which may be realized in many of available technologies as Grid Object Implementation and made accessible over the network as Grid Object Instance

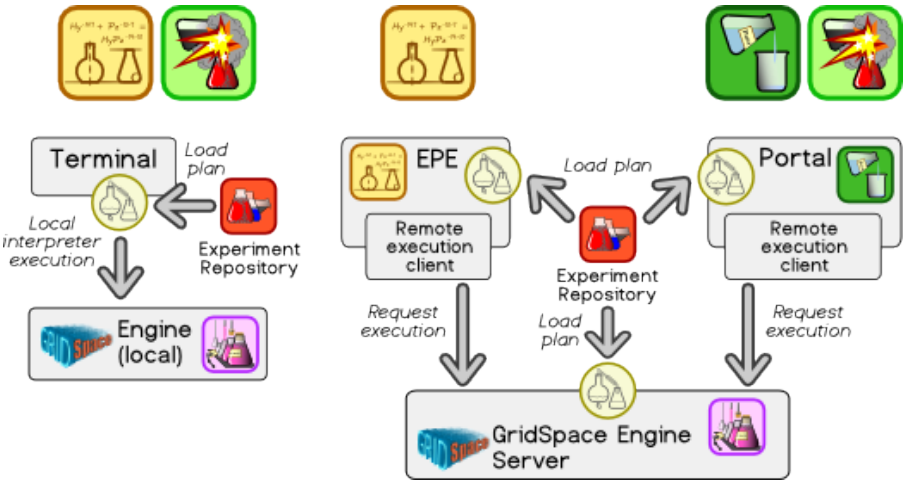


Figure 2.8. Different modes of experiment execution [33]. Local execution requires availability of the *gsel* part of GSEngine, while remote one is run on GSEngine server and requires remote client, *gsec*. Picture comes from the VLvl homepage [12].

User interfaces There are two types of interfaces available in the virtual laboratory [30]:

- ViroLab Experiment Planning Environment (*ViroLabEPE*), which is based on Eclipse platform and supports experiment developers during experiment planning process and allows experiment execution to its users,
- ViroLab Experiment Management Interface (*ViroLab EMI*) - it is a web portal that allows users to run an experiment and manage results of those executions.

Running experiments in virtual laboratory requires using *GSEngine* [25]. The *GSEngine* combines two important parts which are needed in order to run ViroLab experiments. The first one is an interpreter of the JRuby language [6] in which experiments are written. The second part is a runtime library that provides all the specific functionalities of the virtual laboratory. Performing experiments may be done in two ways: locally and remotely [49]. If we choose the local execution, it requires the availability of *gsel* part of *GSEngine*, and it is achievable only from ViroLabEPE or command line. The remote execution is available also through ViroLab Portal (EMI) and requires the availability of remote client - *gsec*. Both types of execution are depicted in Fig. 2.8 and compared in Table 2.3. The selection of experiment running mode may have an influence on the experiment's planning process. It is an important issue especially for a experiment developer, who has to make a decision to which local data access method and user communication API should be used.

Local execution	Remote execution
Requires local GSEngine installation	Relies on a remote GSEngine server
The experiment script is interpreted locally	The experiment is shipped to the remote server
Allows for any kind of GUI interaction with the experiment	Interaction with the experiment is bound with the protocol
Allows for local resource (file) access from the script	Any files needed for execution need to be staged on the server
Requires the user machine to run throughout the execution phase	Allows the user to shut down his/her machine and come back later for results

Table 2.3. Experiment running modes comparison. This table comes from VLvl homepage [12].

2.3. Summary

This Chapter gave a view of a biological background necessary to become acquainted with the idea of bioinformatics experiments. The key facts about cell's information management in DNA molecules, the main process of consuming those information by transcripting DNA to RNA and translating RNA to protein and a concept of heredity was presented. The essential elements of the virtual laboratory: gems and experiments were introduced in the second section of this Chapter. Gems are computational and data access elements, while experiments connect available gems in order to solve a specific problem. The knowledge about the available technologies in which gems may be created will be used to integrate bioinformatics applications in the virtual laboratory. A selected experiment execution mode will influence on the created experiments' structures, described in Chapters 6 and 7.

Chapter 3

Related work

This Chapter gives a short overview of the other approaches to solve a one of the following problems: providing programmatic access to the applications on remote computers, composing different applications in order to get expected results, and creating, executing and managing execution results of the experiments, that are advanced and specialised data flows. In this Chapter the Soaplab and its related programs are introduced as well as the Taverna environment, BIRN network and Meta-PP service.

3.1. Soaplab - SOAP-based Analysis Web Service

Soaplab[1] [26] is a tool that can automatically generate and deploy Web services on the top of existing command-line analysis programs. It is especially well suited for applications with well described input and output parameters. It allows to integrate many applications within a single programming interface.

Soaplab provides three types of Web services:

- Analysis Factory Service - produces a list of available analyses, and provides pointers to them. It is not necessary to use this service - analysis services may be accessed directly
- Analysis Service - A Web service representing a remote analysis, a remote application. All services of this type have the same interface. Parameters are passed by using weakly-typed name-value pairs, and results are obtained in the same way
- Derived Analysis Service - A Web service representing the same remote analysis as the one above, but with specialized interface (strongly-typed methods)

[1] This section is based on http://soaplab.sourceforge.net/soaplab1/

Soaplab uses package AppLab to access to remote applications. Soaplab can wrap as a Web service (almost) any command-line tool, starting from the common UNIX tools. To do this the provider needs to prepare an ACD file which is an application descriptor. This file contains information about a way of application running, its parameters and created results. When a file is created some steps taken by Soaplab must be performed: conversion of ACD file to XML format, the start of AppLab and Tomcat servers, creation and deployment of the Web service. When all those steps are taken, local application is available and can be accessed over the network.

Gowlab

Gowlab[2] is a Soaplab extension used to access programmatically to web pages' resources. Gowlab-service provider needs to create a metadata description of the web resource (in ACD format) and create a Java class extracting useful information from the web page (if page is not simple enough, that default one could not be used). Gowlab uses HTML parsers to analyze the web page, but the provider needs to describe all HTML form elements that have to be filled before retrieving data by GET or POST command.

AppLab

AppLab[3] is a CORBA interface to the command-line driven programs, an implementation of the "Biomolecular Sequence Analysis" engine specification. AppLab uses CORBA communication between a computer executing a command-line program, and the clients' computers. It uses IDL, which allows to invoke remote programs, control them, and deliver back their results. The advantages of AppLab lie in a standard way of analyses description and their input and output data by an XML-based metadata description. This approach makes it possible to plug-in new applications without any additional programming.

Soaplab2

Soaplab2[4] [63] is a new project that enhances the standard Soaplab. It has the same functionality and goals to achieve, but some additional work have to be done, such as implementing plugin architecture, Java usage, CORBA dependencies removal.

3.2. Taverna

The Taverna[5] [38, 55] workbench is one of the most popular pieces of the software that has been produced as a part of the *myGrid* project. This system allows users to design and execute workflows. Each workflow is constructed from one or more

[2] http://soaplab.sourceforge.net/soaplab1/Gowlab.html
[3] http://www.ebi.ac.uk/~senger/applab/
[4] http://soaplab.sourceforge.net/soaplab2/
[5] http://www.mygrid.org.uk/tools/taverna/

services, which may be connected directly with another service. This enables a chain of services to be linked together in a systematic manner. Taverna can use services in many technologies: Soaplab published, Biomoby, WSDL described and others. In this approach services act as components, which are reusable. Taverna users can adds new services by providing WSDL file or location to such file. Workflows are defined as XML files and may be shared over the Internet. Additionally, local resources may be used, such as R language interpreter.

During the workflow creation process, all computation elements are called processors. The processors in Taverna are the primary components of the workflow. They are the entities responsible for both representing and ultimately invoking the tasks from which the workflow is comprised. A processor defines inputs and outputs ports. The port nodes under each processor node are used to connect the outputs of processors either to the inputs of other processors or to workflow outputs. Each service used in the workflow is assigned to one processor. Inputs and outputs may have assigned MIME type descriptions. Some processor types may expose additional editors, that allows to set specific processor's attributes. Each processor in the workflow has its own settings for fault tolerance. These settings include the ability to retry after failures and to fall back to an alternative or set of alternative processors when all retries have been exceeded. Workflow running may be aborted if any of invocation have failed or may be continued with no invocation of "downstream" processors of the failed one.

Taverna is a graphical environment to create, share and run workflows, presented in Fig. 3.1 . All work can be done by mouse-click actions. Workbench is consisted of some parts:

- *Advanced Model Explorer (AME)* - The main section of the AME shows a tabular view of the entities within the workflow. This includes all processors, workflow inputs and outputs, data connections and coordination links. Some items may be expanded to show properties of child items such as alternate processors and input and output ports on individual processors.
- *Interactive Diagram* - This component provides a mechanism for building workflows by interacting directly with a graphical representation of the workflow model.
- *Workflow Diagram* - This component provides a read only configurable view of the workflow in graphical form. Workflow inputs, outputs and processors appear as coloured boxes with arrows between them to represent data and control links.
- *Available services* - This component provides facilities to manage the various services available to the workflow designer and allows the user to manage service libraries, create instances of a service in the form of a processor within the workflow, search services and other such functionality.
- *Results browser* - This window shows the progress of the workflow and also the results on completion.

The Taverna workbench capabilities may be enhanced by using Service Provider Interface (SPI) and adding plugins.

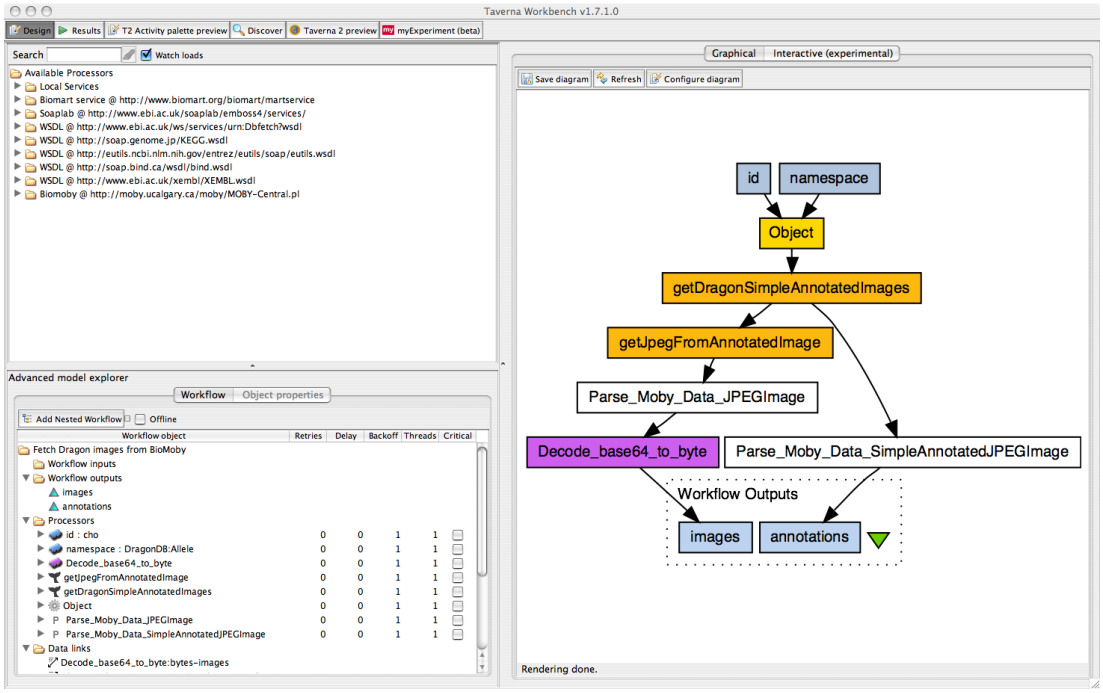


Figure 3.1. Taverna workbench with Advanced Model Explorer (AME), Interactive Diagram for sample Workflow and Available Services windows [38].

3.3. Biomedical Informatics Research Network

The Biomedical Informatics Research Network (BIRN)[6] is a test bed for development of hardware, software, and protocols to effectively share and mine data in a site-independent manner for both basic and clinical research. It currently comprises 3 brain imaging research test beds: Mouse, Human Morphometry and Function. BIRN is a geographically distributed virtual community of shared resources offering tremendous potential to advance the diagnosis and treatment of disease. BIRN enhances the scientific discoveries of biomedical scientists and clinical researchers across research disciplines. BIRN also:

- hosts a collaborative environment rich with tools that permit uniform access to hundreds of researchers, enabling cooperation on multi-institutional investigations.
- synchronizes developments in wide area networking, multiple data sources, and distributed computing.
- designs, tests, and releases new integrative software tools that enable researchers to pose questions and share knowledge across multiple animal models (mouse, human, and non-human primate).

[6] http://www.nbirn.net/

3.3.1. BIRN Coordinating Center

The BIRN Coordinating Center (BIRN-CC)[7] develops, implements and supports the information technology infrastructure necessary to achieve distributed collaborations and data sharing among the test bed participants. The development of flexible approaches to data integration, in conjunction with data modelling and database development, is principal to the BIRN efforts. Other duties span research, dissemination and deployment of the overall system architecture, adaptation of existing hardware infrastructure and software, expansion and support of test bed-specific tools, development of novel software techniques, facilitation of test bed group communication, and daily operation and monitoring of the distributed BIRN hardware.

3.3.2. BIRN Data Repository

To further promote a collaborative research environment, the BIRN has undertaken the development of the public BIRN Data Repository (BDR)[8]. This sustainable archive supports the sharing and exchange of biomedical research data, accepts data generated by the biomedical research community and makes these data freely available, towards fulfilment of NIH guidelines.

3.3.3. BIRN Test Beds

Function BIRN

The FBIRN Test Bed is developing tools to make multi-site functional MRI studies a common research practice. The FBIRN has developed a suite of tools and methods to calibrate and collect data across diverse environments; store and manage the imaging and clinical data in standard ways; analyze multi-site data; and share the results in a permanent location.

Brain Morphometry BIRN

BIRN's Morphometry Test Bed focuses on pooling and analyzing data across neuroimaging sites for potential relationships between anatomical differences and specific memory dysfunctions, such as depression, mild Alzheimer's disease, and mild cognitive impairment. By acquiring and pooling patient data across sites, scientists are analyzing data from a large group of subjects, in order to investigate what structural brain differences correlate to symptoms of memory dysfunction or depression.

Mouse BIRN

The Mouse BIRN Test Bed is helping scientists study neurodegenerative diseases by marshaling multi-modal data from the mouse, as an animal model, to investigate neurological disorders. Researchers across six groups are pooling and analyzing

[7] http://www-calit2.nbirn.net/research/bcc/birncc_ci_primer.shtm
[8] http://www-calit2.nbirn.net/bdr/index.shtm

multi-scale structural and functional data and integrating it with genomic and gene expression data acquired from the mouse brain.

3.4. META-PredictProtein

META-PP is a part of the PredictProtein [58] service, an automatic service for protein database searches and the prediction of aspects of protein structure. META-PP provides a single-page interface to various World Wide Web services for sequence analysis. 'Single-page interface' means that sequence providing is required only once, and any number of a list of services may be used by researcher. Computation results are returned via e-mail. The following features of sequence analysis are covered by META-PP: signal peptides; cleavage sites; O-glycosylation sites; cleavage sites of picornaviral proteases; chloroplast transit peptides and cleavage sites; secondary structure prediction; membrane helix prediction; threading, or remote homology modelling (searching for proteins of known 3D structure that appear structurally similar to your protein); database searches; homology modelling (prediction of protein 3D structure by homology to a sequence similar protein of known structure). A complete list of available services is presented in Meta-PredictProtein documentation[9]. Service accepts sequence or alignment as input in one of the following formats: FASTA (recommended), SWISS-PROT, GCG, PIR, DSSP, SAF, MSF, HSSP.

3.5. Summary

This Chapter presented some examples of the related work. Others' experience and developed solutions will be used during the integration of existing services. In particular, we follow the approach of *SoapLab* for providing command-line applications as remote accessibly Web services and we create services that are facades to the other www services as it is done in *Meta-PredictProtein* project.

[9] http://www.predictprotein.org/doc/explain_meta.html#list

Chapter 4

Analysis of Bioinformatics Applications and Gems

In this Chapter we analyse the structure of the bioinformatics experiments and we provide an overview of the selected applications which have been made accessible in the virtual laboratory. Two types of classifications of the applications are demonstrated in the second section: the first one is based on the integration technology to wrap applications, the other relies on the application scope. Finally, additional mechanisms developed in this thesis, designed in order to make proper integration of the selected and future applications, are shown.

4.1. Model of Bioinformatics Applications

In this section the overall approach to bioinformatics software usage is presented. Then the model of a general experiment is shown. Finally, selected bioinformatics tools integrated in the virtual laboratory are listed.

Biological data are now easily accessible [15] to the scientific community thanks to the rapid development of specialist databases and data querying systems. The next challenge is to provide programs and algorithms that would allow the researchers to extract meaningful information from a mass of initial data. Bioinformatics software is designed to face this challenge. While designing these tools some factors must be taken into consideration ([47]):

- The biologists as end users may not be familiarized enough with computer technology, programing techniques and languages, etc.;
- Software tools must be made available over the Internet to give the global distribution within the scientific research community;

- Programs that solve complex problems have to be provided with support infrastructure to run those programs in a reasonable time.

As it was presented earlier, available bioinformatics software is made in many different technologies. It is often necessary to use more than one application in order to gain results in a single experiment.

We define **metaservice** as a framework that provides access to computational services in unified way (under well defined interfaces). An analysis of bioinformatics software used by researchers entailed a selection of the applications. After that, a set of bioinformatics tools, in which many of the available programs are designed to solve similar problems, was integrated into the ViroLab virtual laboratory and is now available as a set of gems. The gems which are wrappers for software are often available under the same interface, so the researchers can use the virtual laboratory as a metaservice. Thanks to this abstraction of the integrated applications users can easily create new experiments in which available software may be utilized in almost unlimited sequence and if there are more programs to solve this same type of problem, the user can easily select between them and swap selection if needed.

Performing experiments in the virtual laboratory allows researchers to manage the results of those experiments easier, not only final results, but even those obtained during experiment execution. Application programming interface provided in the virtual laboratory gives methods to store results in web repositories as well as locally. In order to enable a connection between any of the available gems it is necessary to provide some converters between formats used by bioinformatics software and libraries. A detailed data formats description is given in section 5.1.2.

4.1.1. A layered structure of the bioinformatics problems

An analysis of the bioinformatics experiments process, carried out in cooperation with virtual laboratory users in the scope of this thesis, contributed to a production of a general bioinformatics experiment model, which is depicted in Fig. 4.1. The main part of this process is a specialized software running aiming a problem or some kind of its sub-problems. To run any of the bioinformatics applications it is required to provide data on which this program would run. Those data may be received from many sources: researcher may put it directly, it can be stored in a locally available file system or accumulated in publicly available databases and obtained by providing its identifiers, or received by using general database querying systems.

Data may be obtained in different format from the one accepted by used application, so that it is sometimes necessary to use format converters. A researcher can do initial data analysis in addition to the main process. Those analyses may be related to obtaining some statistical information, creating new knowledge with data mining or manipulating data by using a clustering technique or dimensionality reduction. The results created during the experiment execution are an input to the next program in the sequence but may be also analyzed in another experiment, so it may be necessary to store those data. The step with bioinformatics software execution is repeated so that the new software also needs input data, which may

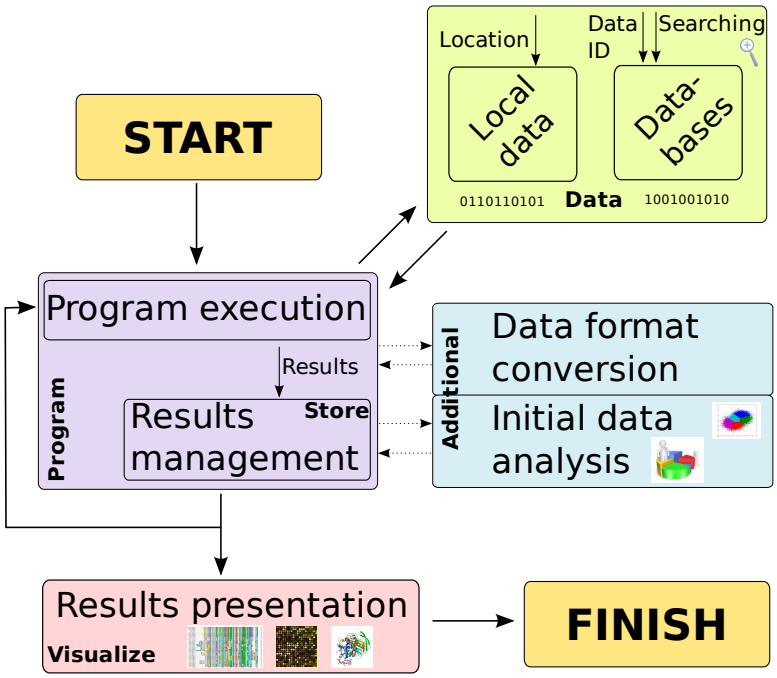


Figure 4.1. General model of bioinformatics experiment process. The main parts are: a specialized software running and results management. Data may be obtained from databases or local files, additional analysis and results presentation is also possible.

require a conversion to an appropriate format or doing an additional analysis. Any number of applications may be used during experiment execution. The final step, which is done when all the results are obtained, is visualization of the results. This step is not necessary to get complete results, but it is often very helpful in their understanding and verification.

4.1.2. Selected bioinformatics software - applications and libraries

The model of the general experiment process and the virtual laboratory users research areas have a great influence on the composition of a provided set of bioinformatics tools in the virtual laboratory. A complete list of those tools, divided according to the types of problems they can solve, is presented below:

- Sequence alignment - a set of tools that search the best multiple sequence alignment and tries to find a way of arranging the sequences of DNA, RNA, or protein to identify regions of similarity that may be a consequence of functional, structural, or evolutionary relationships between the sequences. Available applications are: *ClustalW*, *ClustalW2*, *Muscle*, *T-Coffee*;
- Structure alignment - applications that create multiple sequence alignment which is based on the best alignment of three–dimensional proteins structure alignment. Available applications are: *Mammoth*, *MultiProt*, *SSM*. For aligning only two structures there is also *Dali* software available;

- Protein structure prediction - modelling of proteins folding. *EarlyFolding* is the available application;
- Ligand binding site prediction, protein structure and function analysis - it is a set of applications based on publicly available WWW portals. In the virtual laboratory the following applications are provided: *CastP*, *ConSurf*, *Fod*, *Ligsite*, *Pass*, *PocketFinder*, *QSiteFinder*, *SuMo* and *WebFeature*;
- Statistical analysis - running scripts that are written in *R* language;
- Data clustering - a set of libraries and programs that allow users to perform data clustering. The following are available: *Cluto*, *Cluster 3.0*, *Weka*;
- Microarray data clustering and analysis - it is an own implementation of algorithms used to create and analyze datasets from publicly available general data;
- Data mining - means discovering new knowledge from data by using classifiers, associators and clusterers in *Weka* library;
- Database searching - plenty of public databases are available thanks to using the provided Web services. Data can be accessed by using data identifiers;
- Results presentation - a set of programs that can visualize the obtained data. Available software are: Jmol (protein structure visualization), JalView (sequence alignment), JTreeView (microarray data visualization) and Gnuplot (numerical data visualization).

4.1.3. Summary

The main bioinformatics research areas were discussed in this section. On the basis of this analysis the general structure of bioinformatics experiments was developed and a selection of applications to integrate into the virtual laboratory was performed. Those applications and the way how they were integrated will be described in following Chapters.

4.2. Classification of Gems

Bioinformatics applications may be divided into a few different categories. In this section two types of classifications are presented: the one based on used application technology and the second one based on the scope of usage. Before that, the technologies that are used to integrate bioinformatics software are depicted.

Gem is another name of Grid Object abstraction, and from now on we will be using both this terms interchangeably. New gems are added to the virtual laboratory in order to enhance its capabilities. As it was presented in Table 2.2, gems can be created in several technologies. A common part of the gem creation process in any technology is defining the interface which the gem will realize. Bioinformatics applications are integrated in the virtual laboratory using three of the available technologies. Two of them, Web services and Mocca components, were presented in Table 2.2 and the last one - Local Gem has not been introduced yet.

Web service

Web service is defined by the W3C as "a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards."[1]. Web services are often remotely accessible API used to run local programs, as it was presented in section 3.1. Web service can be accessible by:

- a defined descriptor in Web Services Description Language (WSDL),
- publishing and searching available registries with well defined protocol (e.g. UDDI registers),
- calling methods from a known service interface,
- using other services to access them - it may be a part of another service or it may use other services.

Bioinformatics applications are integrated by using two of those types of Web services: the one with WSDL description and the one with well defined interfaces, which is done by defining Grid Object Class in Grid Resource Registry. Communication with Web service is made by sending messages in SOAP protocol. Another type of Web services - REST (REpresentational State Transfer) - is often used to provide resources - by well defined URI identifiers. All resources are accessible in a unified way, and are easy to search and download, thanks to the structure of URI address.

Mocca components

Mocca [51] is a CCA compliant distributed component framework based on H2O platform. Mocca allows building component applications on the distributed resources available through H2O. H2O [4] is a Java-based middleware platform for building and deploying the distributed applications. Current Mocca version, called Mocca_Light, is a pure-Java implementation of the CCA specification.

Local Gem

Local gems are ViroLab experiments scripts that undertake some precisely defined tasks. Those experiments can be used in other, larger experiment scripts as simply as standard gems. There is a possibility to use virtual laboratory API as well as other Grid Objects inside a local gem body. Local gem functionality must be defined as a JRuby class. All methods in this class must have the same signatures as those defined in Grid Object class to which Local Gem implementation is assigned. Detailed information about Local gem requirements are presented in Table 4.2.

Adding new functionality to virtual laboratory requires three steps to be taken. Firstly, it is essential to register Grid Object class, which defines the interface. Then this class has to be implemented in one of the available technologies and the detailed information about this implementation must be also registered in GRR.

[1] http://www.w3.org/TR/ws-gloss/

Parameter	Description	Is required?
Package	Name of package where gem will be available in the registry. Packages provide separated namespaces	Yes
Name	Gem name. Name and package constitute gem identifier which is used during gem creation in experiment execution process	Yes
Description	Description of gem, its capabilities, references to external sources, etc.	No
Operation signatures	Gem interface, list of available methods with required parameters and returned results descriptions.	At least one

Table 4.1. The parameters of a new gem registration

Technology	Gem Implementation	
Web service	Name	Implementation name
	Type	Web service type (RPC, DOCUMENT)
	Namespace	Required only if no WSDL file is available
MOCCA	Name	Implementation name
	Port name	Mocca port name
	Port class	Mocca port class that defines port interface
	Component class name	Mocca component class name - a full Java path
	Component classpath	Address to public component class implementation (usually an URL to JAR file)
Local Gem	Name	Implementation name
	Local gem type	Gem type, only LOCAL type was used
	Script	Script file with gem definition. Whole code must be enclosed in class definition, which name has to be the same as the Grid Object class name, and all method signatures must be the same as Grid Object class interface

Technology	Gem Instance	
Web service	Name	Instance name
	Endpoint	Web service endpoint address - it may be direct Web service port or URL address to WSDL file
MOCCA	Name	Instance name
	Endpoint	H2O kernel location where component should be deployed
Local Gem	Name	Instance name, instance must be defined but it is not used

Table 4.2. Required parameters during registration process for different gem technologies.

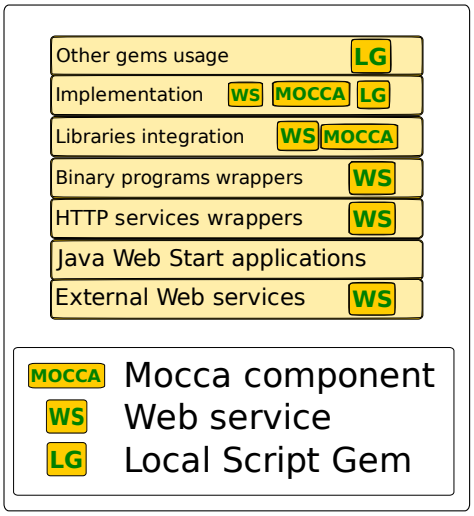


Figure 4.2. Layers of the gem technology with relevant Grid Object implementation. Three main implementation types are used (Mocca components, Web service and Local Gem) to integrate bioinformatics applications availble as Web services, WWW services, binary programs (also JAR files), libraries and implemented from scratch in the scope of this thesis.

Finally, implementation must be published and registered as Instance in GRR. The first step is independent of technology. Information needed to register Grid Object class are listed in Table 4.1. Despite the fact that all the gems are accessed in a uniform way from the experiment, GSEngine requires specialized information to execute gem methods in different technologies. All information necessary to register the gem created in any of the used technologies are listed in Table 4.2.

4.2.1. Technology layers

Three technologies used to integrate bioinformatics software in connection with the type of application can be ordered and divided in seven categories. This division is presented in Fig. 4.2 and described in this section.

External Web services. Integration is based on registering services interfaces, which are created and maintained by an external provider. Services are often described by WSDL descriptor. Example gems: *ClustalW*, *T-Coffee*.

Java Web Start applications. Binary Java applications with Graphical User Interface are provided with Java Web Start technology. Especially, applications for data visualization are integrated with this technology usage, which requires a generation of the JNLP files with the application description.

HTTP services wrappers. Methods that solve bioinformatics problems are often published as WWW portals and it is the only way to use those algorithms. In order

to use such Web-based services in the virtual laboratory, where services may be used massively, it is necessary to wrap communication with WWW portals. Portals are human-oriented and to take an automation step WWW pages have to be annotated with the names of the used fields, as it was presented in 3.1. Communication is made mainly with POST and GET methods from HTTP protocol. To support some WWW services not being ready to handle massive requests and parallel analyses of many tasks, a queuing system, described in section 4.3.1, was created. Some examples of the gems are: *CastP*, *ConSurf*, *PocketFinder*.

Binary program wrapping. Running binary programs locally often requires that their input data should be stored as a file in a local file system, and their options have to be passed as a command line string or specially prepared parameters file. Their outputs are available on a standard output or as a result file stored locally. A standard error output should be also observed and analyzed in order to find out whether the program has finished successfully. The process of integration of this type of applications is described more precisely in section 4.3.2. The examples of the gems implemented in this technology are: *Fod*, *Pass*, *Mammoth*, *Gnuplot*, *R_gem*.

Libraries integration. Libraries integration is possible only if a new application is created in the same programming language as the library. It requires recognition which library function should be available in the virtual laboratory; additionally, a specialized programming code that prepares data and runs library function must be created. Example gems wrapping Java library as Mocca components are: *WekaClassifier*, *WekaClusterer*.

Implementation. This layer is constituted by the gems that were developed from scratch to provide special functionality, do not use external programs, and are not wrappers for the specialized libraries. The examples of implemented gems are: *GeoDataProvider*, *ClustalWUtils*.

Other gems usage. The prepared gems are used to provide new functionality which can be applied in many experiments. They are created as JRuby classes and can use GSEngine API. The examples of such gems implemented as Local Gems are: *WekaClusteringInterface*, *ScopDb*.

4.2.2. Usage scope layers

Bioinformatics processes have a quite well defined structure, as it was depicted in section 4.1 and in Fig. 4.1. According to this structure bioinformatics applications can be divided into four usage layers, illustrated in Fig. 4.3. Those layers are examined in this section.

Database access layer

Each bioinformatics experiment requires input data. Those data may be created and supplied by a researcher, which means that one needs to put the data directly

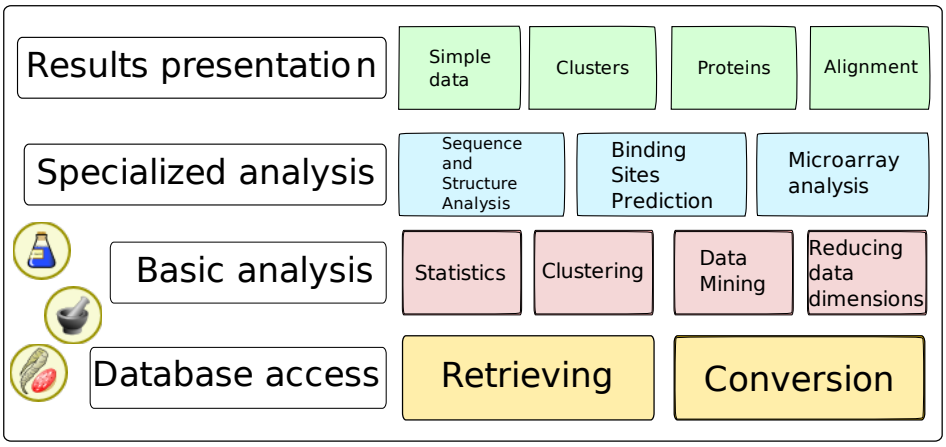


Figure 4.3. Layers of the gem scope of usage. Four main layers are recognized: *Bioinformatics Database access*, for retrieving data and converting file formats to the one accepted by appropriate software; *Basic analysis*, used in a variety of experiments, where statistical analysis, data clustering, data mining and data dimensions reducing gems are available; *Specialized analysis* used to solve specific type of problem and *Results presentation* for visualizing data.

into an experiment code or may be retrieved from specialized databases based on other information, such as data identifiers. It is often required to parse and format the retrieved data in order to analyze them in an appropriate program or library. Gems belonging to this layer are presented in Fig. 4.4 and detailed in section 5.1.

Basic analysis layer

Basic analysis gems can be used in a variety of bioinformatics experiments. As we can see in Fig. 4.5, the following gems belong to this layer: statistical analysis (described in section 5.2), data mining analysis (section 5.3), data clustering (section 5.4) and reducing data dimensionality (section 5.5). This layer is called "basic" because the gems can be used in any type of bioinformatics process, not only in specialized research.

Specialized analysis layer

It contains gems that are used to solve specific types of problems. In the scope of this thesis the gems were created to perform: sequence and structure comparison for protein families (Chapter 6), comparing different methods and services in ligand binding site prediction (Chapter 7) and performing microarray data analysis (Chapter 8).

Results presentation layer

This layer groups all the gems that can be used to visualize the results of the experiment performance. It contains especially Jmol, JalView, JTreeView and Gnu-

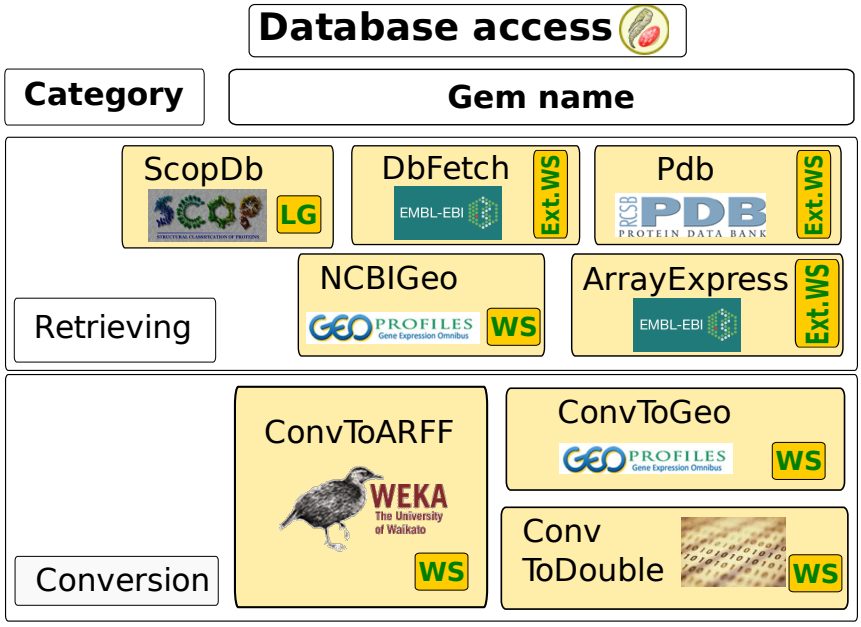


Figure 4.4. *Bioinformatics database access* layer. Two sub-layers are: Data retrieving, with gems that access various databases and Format Conversion for adjusting data type to one of the following formats: ARFF file, Geo Object model and Array of double.

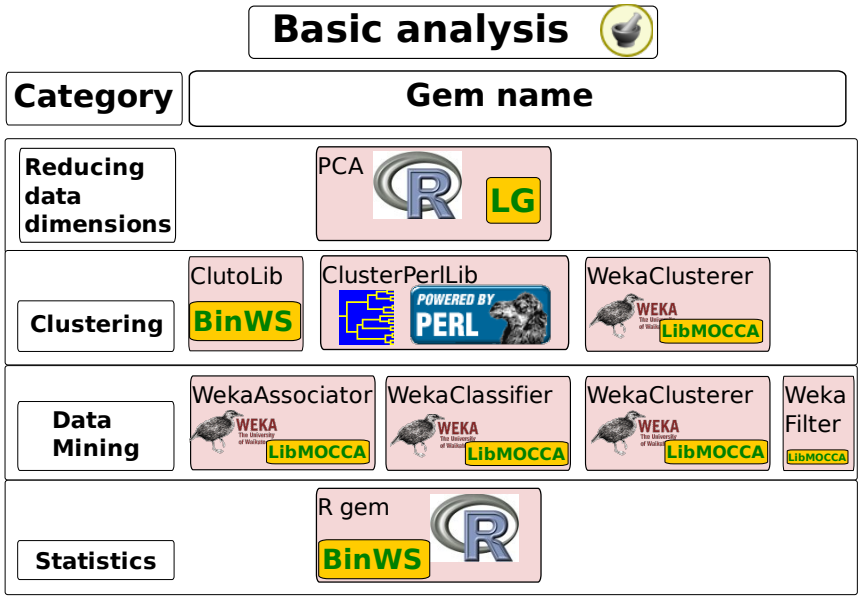


Figure 4.5. Basic analysis layer. Gems from this layer can be used in a variety of bioinformatics experiments. Four main categories are covered: Statistics (R), Data mining (Weka), Data clustering (Cluto, Cluster, Weka) and Reducing data dimension (PCA, MDS in R).

plot programs. A detailed description of the available visualization techniques is provided in Chapter 9.

4.2.3. Summary

The aim of this section was to develop a classification of the integrated gems in comparison to previous section, where applications classification and selection were discussed. Integrated gems have been divided in categories by taking gem's development technology into consideration. The second categorization was performed on the basis of gem's application in the research areas.

4.3. Additional Integration Mechanisms

In order to ensure that the whole bioinformatics software will work correctly in the virtual laboratory some additional mechanisms were developed in the scope of this thesis, in addition to the already existing gem technologies. Thanks to this work, adding new applications that are technologically similar to the existing gems, is easier. In this section two mechanisms are presented: a tasks queuing system used when external service is not multitasking enough, and the architecture of binary program wrapping.

4.3.1. Tasks queuing system

Bioinformatics services are often Web-based and user oriented. Due to their architecture and to limitation on available computational resources, these services often do not allow running many tasks simultaneously, so that tasks have to be queued before sending them to the service. On the other hand communication with Web services in SOAP protocol is limited by timeouts. However, the Web services are often used as the virtual laboratory gems. To overcome these problems, an architecture of the task queueing system that provides asynchronous computation methods was designed. This system has been primarily developed for an integration of web-based services doing ligand binding sites prediction (Chapter 7), but its solution is general enough to use it in other gems where tasks may require queuing.

General architecture of the system that queues and executes computational tasks is presented in Fig. 4.6. It consists of a set of general classes, some of which have to be specialized in order to make the system work properly and obtain a concrete problem solution. A detailed description of each class type is presented in Table 4.3.

The entry point to the system is a *Service* class. It defines external system interface. The object of this class also defines system parameters, such as its name, a maximal number of tasks analyzed in parallel and maximal processing time available for each task. The *Service* object creates also an instance of *TaskAnalyzer* which is then replicated and maintained by *TaskManager*. Service object is a Facade (design pattern, [31]) to task manager. Each computation request is supplied with required information and passed to *TaskManager*. Manager creates new *Task* object, gives an

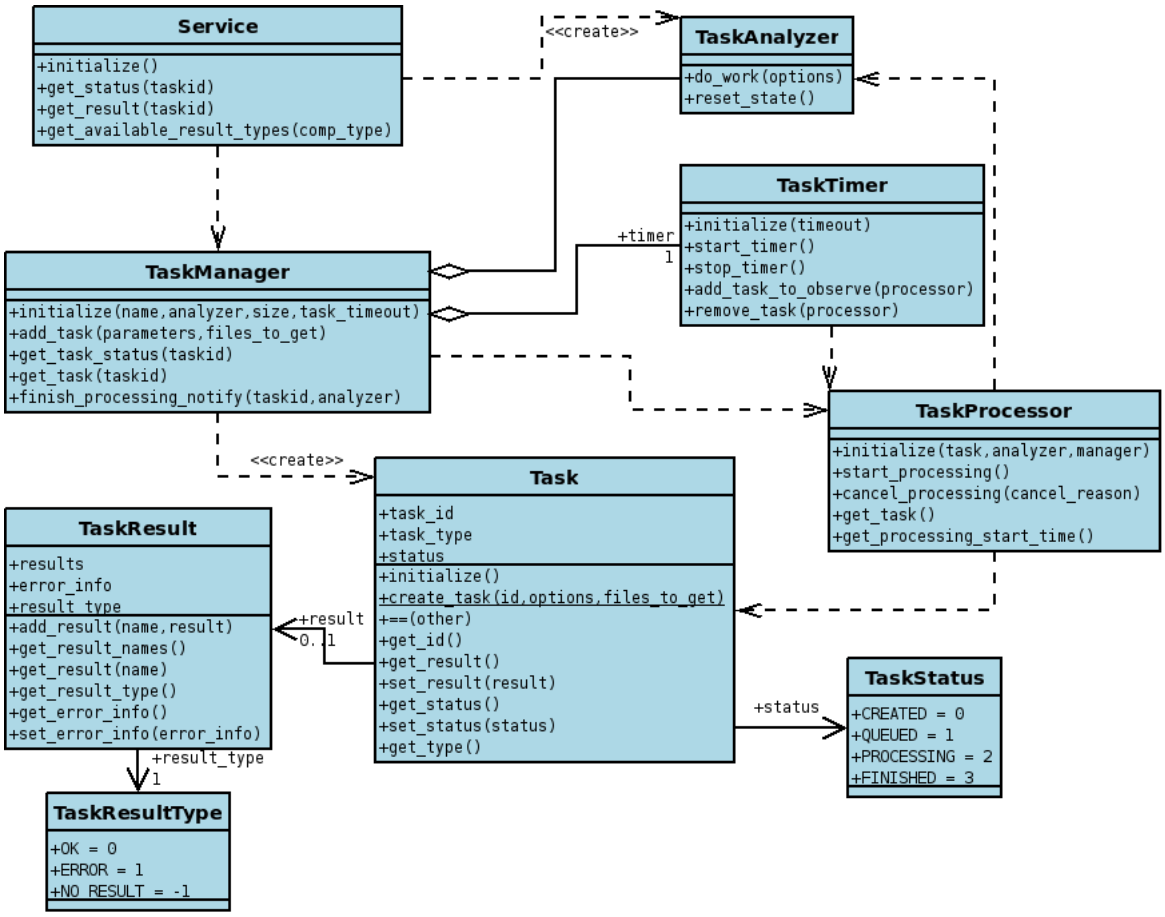


Figure 4.6. General architecture of the tasks queueing system. Service class defines system interface. Task manager keeps tasks queue, finished tasks list and starts task analysis process in new TaskProcessor. Analysis is performed by TaskAnalyzer.

Class	Type	Responsibilities
Service	Specialization required	System interface
Task	Specialization possible but not required	Task description, parameters and processing results
TaskAnalyzer	Specialization required	Task analyzing, communication with analyzer service
TaskManager	General	Managing tasks, creating, adding, running and returning results
TaskProcessor	General	Starts tasks analyze in separate thread
TaskResult	General	Analyzing results or error information
TaskResultType	General	Result type - Success or Error
TaskStatus	General	Task status
TaskTimer	General	Observes currently processing tasks, if processor has reached time limit analysis is stopped

Table 4.3. Tasks queuing system classes.

identifier as well as options for it and creates a list of result types that are expected to be retrieved. Task creation is a static method in *Task* class. The created task is then put at the end of a task queue, and *QUEUED* status is assigned to it. When the manager object is created, task analyzer objects are initialized. The number of analyzers is a parameter passed to the manager from the *Service* object. When a new task is added to the tasks queue, the manager checks an available analyzers list. If there is at least one *TaskAnalyzer* object, the manager removes it from the list, creates a new processor for the task (*TaskProcessor*), removes the first task from the queue and then with those three objects the task processing is started. A processor thread is also added to the *TaskTimer* observer, which ensures that if the task is not finished until the maximal time, task analysis will be stopped. The processor gets an option list from *Task* object, and then calls analyze method on *TaskAnalyzer*. When the analysis is finished, the processor gets results (*TaskResult* object) from the analyzer and puts it into the *Task*. If the analysis fails, the processor creates new *TaskResult* with information about error reason (*TaskResultType* is set to *Error*). When the processing is finished *TaskProcessor* notifies the manager about it, the processor is then removed from the timer observer, the task is put into the finished tasks queue and manager the tries to run a next task if there is any waiting. As *TaskAnalyzer* is responsible for performing the complete analyzing process and for the production of the results, it must be always specialized for a concrete system.

System management

A *Service* class realizes an additional management interface *IManagable*. Methods from this interface are responsible for managing the service, the tasks queues and for providing information about the service state. Each method requires passing a user name and a password as a first parameter. These values must be passed as a

single string, separated with ':' ('user:password'). Before the method execution it is compared to the values defined in the text file *users.pwd,* which have to be placed at the same directory where *Service* has been run. If no according user:password string has been found in this file an exception is thrown. The management methods are not available through the virtual laboratory by default, but can be accessed only from the locally placed client (from the place where *Service* has been run). The following methods and types are defined for system managing:

- `ParamType` - Enumerate type with parameters names, available values are: *name*, *processors*, *timeout*;
- `ListType` - Enumerate type with tasks lists names, available values are: *queued*, *processing*, *finished*;
- `get_parameters` - returns a map with all the parameters and their values that are currently set in the service;
- `set_parameter(param_type, value)` - set selected parameter from one available in *ParamType* definition, this method allows the manager to redefine the number of parallel processing tasks or change the timeout for task processing;
- `get_queue_length` - returns a length of the list of the tasks that are waiting for computing;
- `get_tasks_ids(list_type)` - get all task identifiers from the selected list, as a result a map with task identifiers as a key and task information (such as task status and task creation time) as a value is returned;
- `remove_task(list_type, task_id)` - remove the selected task from the selected list. If no task is available an exception is thrown. When the task is removed from currently processing tasks, the analysis process is stopped, an information on the reason of processing cancellation is set in TaskResult object and a Task is moved to the finished tasks list;
- `remove_all_tasks(list_type)` - remove all tasks from the selected list;
- `reset_service` - clear all the lists, remove all currently created processors and reset Service properties to the default ones.

The presented solution is a generic framework. A concrete realization and usage of this system is presented in section 7.3.

4.3.2. Binary program wrapping

Locally available applications have some common features that are present when we try to run them from other applications. An frequent requirement is that program input data should be placed in locally stored files, produced results are often available as files on disk and finally analyzing standard output and standard error output is required. Most of the programs can be configured by passing command line options or preparing specified configuration file. In order to guarantee an extensible binary application integration mechanism, *BinaryProgramWrapper* module was developed. This module is responsible for (activities are listed in order):

- Creating an unique temporary directory for storing data. Directory name depends on time and parameters passed to the module;

- Creating and storing files with input data. Those files should have correct names;
- Options preparing as a command line string or parameters file. An appropriate object, that is specialized for handling program options, is injected into the module;
- Running external program. Program name and location are configurable;
- Standard output and standard error output analysis. If any program requires additional output analysis - a specialized object may be injected into the module and this task is then redirected to this object;
- Reading output files. File names and localizations are passed to the module when a method is called;
- Removing the temporary directory and returning results.

This module was used to integrate ligand binding site prediction applications, such as *Fod* and *Pass* or protein structure alignment applications, such as *Mammoth* and *MultiProt*.

4.3.3. Summary of extension mechanisms

The general mechanisms which were developed to support an integration process were described in this section. We presented an architecture to tasks submission, running and managing results of their execution that will be used to integrate the services for prediction of ligand binding sites in protein (see Chapter 7) and general command-line program wrapper that will be used to integrate for example *Fod* and *Pass* programs (also described in Chapter 7).

4.4. Summary

An analysis of the process of bioinformatics experiments have led to the production of a general bioinformatics experiment model and to the selection of the applications that were integrated into the virtual laboratory. The model and the list of the selected software were described at the beginning of this Chapter. On the basis of the selected applications, two types of gems' classifications were introduced: the one based on used application technology, where seven different technology layers have been distinguished and the second based on scope of usage, where Database access, Basic analysis, Specialized analysis and Results presentation layers have been found. Finally, two additional mechanisms for integrating applications: the tasks queuing system and the wrapper for binary program, were developed in the scope of this thesis.

Chapter 5

Database Access Layer and Basic Analysis Layer Gems

This Chapter describes gems that can be used in any type of bioinformatics experiment. It starts with the presentation of the available bioinformatics databases, data format description and methods and gems used for obtaining data. Then two packages of data analysis in virtual laboratory are shown: R statistical language and Weka data mining library. Finally, the possibilities of data manipulation with clustering libraries and data dimensionality reduction methods are introduced.

5.1. Bioinformatics Database Access Gems

Publicly available specialist databases are one of the most important factors contributing to the rapid development of bioinformatics science. Thanks to those systems researchers from all over the world can share their results and compare new methods and algorithms in order to obtain better and faster programs. Complex Web-based interfaces allow users to search and manage (by adding or modifying) contents of databases. Data can be accessed in an easy way and the amount of information available in the databases grows exponentially. This section examines databases, file formats and created gems which are used in bioinformatics experiments in the virtual laboratory.

5.1.1. Bioinformatics Databases

Databases accessible from the virtual laboratory experiments which were used in this thesis are described below.

Protein Data Bank

The Protein Data Bank (PDB)[1] [16] archive is the single worldwide repository of information about the 3D structures of large biological molecules, including proteins and nucleic acids. These are the molecules of life that are found in all organisms including bacteria, yeast, plants, flies, other animals, and humans. Understanding the shape of a molecule helps to understand how it works. This knowledge can be used to help deduce a structure's role in human health and disease, and also in drug development. The structures in the archive range from tiny proteins and bits of DNA to complex molecular machines like the ribosome. Data available in Protein Data Bank are stored in pdb files and can be found by passing four-chars identifier *structureId* (also called *pdbid*). Database is equipped with advanced searching system, available also as SOAP Web service. From a plenty of methods provided by this service those which may be particularly interesting and usable in the virtual laboratory are the methods concerning getting amino acid sequence for a selected structure and chain, listing chains in a selected structure, getting the length of chains and querying database with XML queries.

Structural Classification of Proteins

The SCOP database[2] [53], created by manual inspection and abetted by a battery of automated methods, aims to provide a detailed and comprehensive description of the structural and evolutionary relationships between all proteins whose structure is known. As such, it provides a broad survey of all known protein folds, detailed information about the close relatives of any particular protein, and a framework for future research and classification. A description is available as a set of tightly linked hypertext documents which make the large database comprehensible and accessible. In addition, the hypertext pages offer a panoply of representations of proteins, including links to PDB entries, sequences, references, images and interactive display systems. Proteins are classified to reflect both structural and evolutionary relatedness. There are three principal levels in the hierarchy: Family (Clear evolutionarily relationship), Superfamily (Probable common evolutionary origin) and Fold (Major structural similarity). Searching SCOP database is also available from the PDB querying system.

Gene Expression Omnibus

GEO[3] serves as a public repository for a wide range of high-throughput experimental data. These data include single and dual channel microarray-based experiments measuring mRNA, miRNA, genomic DNA (arrayCGH, ChIP-chip, and SNP), and protein abundance, as well as non-array techniques such as serial analysis of gene expression (SAGE), mass spectrometry peptide profiling, and various types

[1] http://www.rcsb.org/pdb/static.do?p=general_information/about_pdb/index.html
[2] http://scop.mrc-lmb.cam.ac.uk/scop/intro.html
[3] http://www.ncbi.nlm.nih.gov/projects/geo/info/overview.html

of quantitative sequence data. The basic record types within the primary database are following:

Platform A *Platform* record defines the list of elements that may be detected and quantified in that experiment (e.g., cDNAs, oligonucleotide probesets). Each *Platform* record is assigned a unique and sTable GEO accession number (GPLxxx).

Sample A *Sample* record describes the conditions under which an individual *Sample* was handled, the manipulations it underwent, and the abundance measurement of each element derived from it. Each *Sample* record is assigned a unique and sTable GEO accession number (GSMxxx). A *Sample* entity must refer to only one *Platform* and may be included in multiple *Series*.

Series A *Series* record links together a group of related *Samples* and provides a focal point and a description of the whole study. *Series* records may also contain Tables describing extracted data, summary conclusions, or analyses. Each *Series* record is assigned a unique and sTable GEO accession number (GSExxx).

DataSet *DataSet* records are assembled by GEO curators. GEO *Series* data are reassembled by GEO staff into GEO *Dataset* records (GDSxxx). A *DataSet* represents a curated collection of biologically and statistically comparable GEO *Samples* and forms the basis of GEO's suite of data display and analysis tools.

Profile *Profiles* are derived from *DataSets*. A *Profile* consists of the expression measurements for an individual gene across all the *Samples* in a *DataSet*.

5.1.2. Formats of bioinformatics data files

Data available in the databases can often be downloaded as a text file. The knowledge about those files structure and organization is necessary for a proper analysis and data modification. This section presents list of the most often used file formats in the bioinformatics experiments.

PDB File

This file describes 3D structure of proteins. It is organized as a 80-column file with some well defined sections. Each line in file is an individual and self-identifying record. The first six columns of every line contain a record name, that is left-justified and separated by a blank. From the available sections we can distinguish a title section, primary and secondary structure description sections and a coordinate section. The last one contains most of the information from the ViroLab gems' and experiments' point of view, because in this section protein atoms positions are defined in ATOM records. Exact ATOM record structure is presented in Table 5.1. All sections and records description are presented in a file format documentation [13] available at *http://www.wwpdb.org/docs.html*.

FASTA Format

FASTA is a text-based format for representing either nucleotide sequences or peptide sequences, in which base pairs or amino acids are represented using single-letter codes. The format also allows sequence names and comments to precede the sequences. A sequence in FASTA format begins with a single-line description, followed

Columns	Data Type	Field	Definition
1 - 6	Record name	'ATOM '	
7 - 11	Integer	serial	Atom serial number
13 - 16	Atom	name	Atom name
17	Character	altLoc	Alternate location indicator
18 - 20	Residue name	resName	Residue name
22	Character	chainID	Chain identifier
23 - 26	Integer	resSeq	Residue sequence number
27	AChar	iCode	Code for insertion of residues
31 - 38	Real(8.3)	x	Orthogonal coordinates for X in Angstroms
39 - 46	Real(8.3)	y	Orthogonal coordinates for Y in Angstroms
47 - 54	Real(8.3)	x	Orthogonal coordinates for Z in Angstroms
55 - 60	Real(6.2)	occupancy	Occupancy
61 - 66	Real(6.2)	tempFactor	Temperature factor
77 - 78	String(2)	element	Element symbol, right-justified
79 - 80	String(2)	charge	Charge on the atom

Table 5.1. Atom record structure in PDB file.

by lines of sequence data. The description line is distinguished from the sequence data by a greater-than ('>') symbol in the first column. The word following the '>' symbol is the identifier of the sequence, and the rest of the line is the description (both are optional). There should be no space between the '>' and the first letter of the identifier. It is recommended that all lines of the text should be shorter than 80 characters. The sequence ends when another line starting with a '>' appears; this indicates the start of another sequence. A sequence header may contain any information, but a set of well described headers is available. Thanks to this header's format the sequence may be linked to data in an appropriate database. The sequence can be written by using standard nucleotide / amino acid description characters and also using some wildcards. A complete list of codes is presented in Table 5.2.

GEO

Microarray experiment data are available in two formats - raw data (RAW) or preprocessed data (SOFT). Files with raw data are specific to the microarray platform which was used in the experiment, e.g. Affymetrix CEL file or GenePix GPR file. Preprocessed data are available in text files. A file with samples set is divided into two parts: description and data. The description part consists of sections. Each section starts with '^' sign and a section name and collects parameters, each in separated line that starts with '!'. Among a plenty of dataset parameters the most important are DATASET section parameters:

dataset_platform - a microarray platform identifier
dataset_feature_count - a number of genes that are analyzed in each sample
dataset_sample_organism - an organism on which experiment was performed

Nucleic Acid Code	**Meaning**	**Amino Acid Code**	**Meaning**
A	Adenosine	A	Alanine
C	Cytosine	B	Aspartic acid or Asparagine
G	Guanine	C	Cysteine
T	Thymidine	D	Aspartic acid
U	Uracil	E	Glutamic acid
R	G A (puRine)	F	Phenylalanine
Y	T C (pYrimidine)	G	Glycine
K	G T (Ketone)	H	Histidine
M	A C (aMino group)	I	Isoleucine
S	G C (Strong interaction)	K	Lysine
W	A T (Weak interaction)	L	Leucine
B	G T C (not A)	M	Methionine
D	G A T (not C)	N	Asparagine
H	A C T (not G)	O	Pyrrolysine
V	G C A (not T, not U)	P	Proline
N	A G C T	Q	Glutamine
X	masked	R	Arginine
-	gap of indeterminate length	S	Serine
		T	Threonine
		U	Selenocysteine
		V	Valine
		W	Tryptophan
		Y	Tyrosine
		Z	Glutamic acid or Glutamine
		X	any
		*	translation stop
		-	gap of indeterminate length

Table 5.2. The complete list of accepted FASTA codes.

dataset_sample_count - a number of separated microarray values (each sample refers to one microarray)
dataset_value_type - a type of preprocessed value

The second part of dataset file contains gene expression level values. This part contains a comment section, where all the samples in dataset are described (lines start with '#' and contain sample identifiers), matrix values with header part (first line are column headers: a gene identifier, a gene name, samples identifiers), and in next lines gene expression levels assigned to the appropriate genes (genes descriptors are provided in the first two columns of each line).

When a user wants to download data for a single sample, resulted file structure is similar to that described above. Important parameters in the description sections are:

Sample_type - a type of preprocessed value
Sample_organism - an organism on which microarray experiment was performed
Sample_platform_id - a microarray platform identifier
Sample_data_row_count -a number of genes that are analyzed in this sample.

Each sample of information is connected to one physically performed microarray experiment. Data section contains at least two columns - gene identifier and expression level, but may be extended with additional two columns - transcript presence and significance level.

A detailed description of a microarray dataset and sample terms is presented in Chapter 8.

5.1.3. Bioinformatics data access gems in the virtual laboratory

Virtual laboratory capabilities may be enhanced by adding new gems. Database access layer gems allow to download data from supported databases. This section lists available gems, that are used in bioinformatics experiments.

PDB

Gem class: `org.pdb.Pdb`

SOAP Web service provided by Research Collaboratory for Structural Bioinformatics (RCSB) Protein Data Bank. In the virtual laboratory four service methods are available:

- `getSequenceForStructureAndChain` - downloading amino acid sequence for structure defined by pdbid identifier and one letter chain code,
- `getChains` - downloading chain codes for selected structure, identified by *pdbid*,
- `getChainLength` - returns chain length (number of residues in chain) for selected protein (*pdbid*) and chain (*chainid*),
- `runXmlQuery` - advanced database queries can be executed. The query must be structuralized in XML format. As a result of query execution, array of *pdb identifiers* is returned.

This gem is used in a protein sequence and structure comparison experiment (presented in section 6.2), where amino acid sequences are downloaded, and as a part of *Scop* gem, where advanced queries are executed.

DbFetch

Gem class: `uk.ac.ebi.Dbfetch`

Web service is provided by European Bioinformatics Institute . *DbFetch* allows users to retrieve entries from various up-to-date biological databases using entry identifiers or accession numbers. Querying databases is done with building query string in "database:identifier" format. In bioinformatics experiments this gem is used to download from PDB database files with protein crystal structure definition. As a database identifier 'PDB' string is used in this case[4]. Additionally, in order to retrieve data, required data format may be defined when queries are being executed. Methods to check available data format and style for every supported database are available.

GeoDataRetriever

Gem class: `cyfronet.gridspace.gem.microarray.GeoDataProviderService`

Gem provides an access to microarray experiment data. A detailed description of this gem is presented in section 8.2.

Scop

Gem class: `cyfronet.gridspace.gem.bioinfo.data.ScopDb`

The Gem is created in Local Gem technology. It uses PDB gem capabilities and REST Web services provided by RCSB institution. Pdb database allows users to search Scop database and get protein *ids* based on protein family identifier. In order to obtain this identifier a special query is prepared and then HTML response is parsed. This identifier is used to create XML query which is then executed on *PDB* gem. As a result of this execution protein pdbid codes array, which belongs to a sought family, is returned. This array may be empty if the queried family was not found. A sequence diagram of the query execution in *ScopDb* gem is presented in Fig. 5.1.

5.1.4. Data format conversion

A variety of bioinformatics software tools available in the virtual laboratory requires additional elements to complete cooperation. These are format converters which adapt data formats used by different programs and libraries. In the virtual laboratory three types of data format may be converted:

ValuesArray Numeric data in double format. This format may be used for any of bioinformatics data;

ARFF Text file used within *Weka* library. Contains header and data sections;

[4] Complete list of accessible databases is available at http://www.ebi.ac.uk/cgi-bin/dbfetch

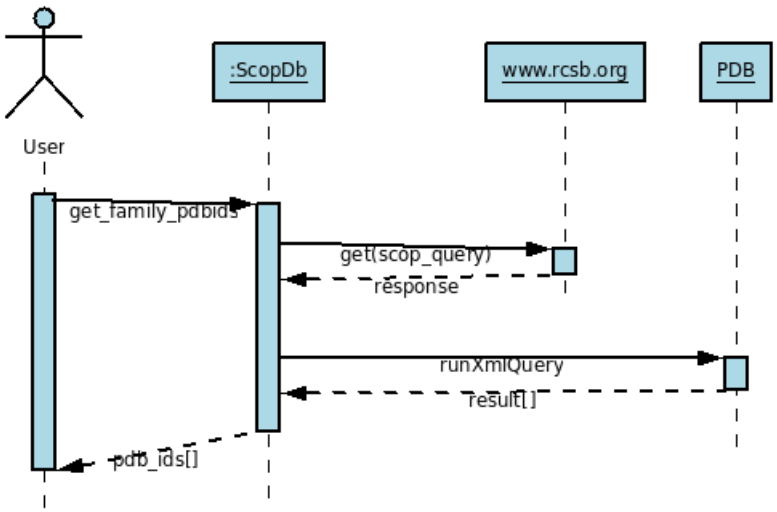


Figure 5.1. ScopDb query execution diagram. At first, family protein identifier is obtained from Web page, then, XML query is prepared and executed on Pdb gem.

GeoDataSet This object is available in microarray gems and experiments. The conversion between these formats is described in Table 5.3.

5.2. Statistical Analysis

In bioinformatics experiments a statistical analysis may be used to obtain simple statistics values, such as correlation between data series. It is also suitable for specialized tasks, such as testing biological hypothesis or deriving knowledge from high-throughput data. In the virtual laboratory possibilities of the statistical analysis have been provided by R package, which is described in this section.

5.2.1. Statistical computing in R

R [8] is a language and environment for statistical computing and graphics. *R* provides a wide variety of statistical (e.g. linear and nonlinear modelling, classical statistical tests, time-series analysis, classification, clustering and many others) and graphical techniques, and is highly extensible. It also provides a variety of plotting options. *R* language is interpreted and object-oriented. One of the main advantages of the *R* is that it can be easily extended via packages. It is a way in which Bioconductor [1] project adds tools for the analysis and comprehension of genomic data into *R* environment. The Bioconductor project aims to provide access to a wide range of powerful statistical and graphical methods for the analysis of genomic data. Analysis packages are available for: pre-processing Affymetrix and cDNA array data; identifying differentially expressed genes; graph theoretical analyses; plotting genomic data. Capabilities of the Bioconductor project are pre-

Input	Output	Description
ARFF	ValuesArray	'@data' part of ARFF file is directly converted, additional information are not required, as a second result an identifiers array with ARFF attributes and relation name is returned
GeoDataSet	ValuesArray	Gene expression level values are returned directly, no additional information is required, additionally map with identifiers 'gds_id', 'gds_platform', 'gds_data_type', 'gds_samples_count', 'gds_features_count' and 'samples' is returned, 'gds_id' is dataset identifier, other parameters were described in section 5.1.2
ValuesArray	ARFF	Values are inserted into @data section, as a second parameter array with attributes names is required to fill @attribute part of file
GeoDataSet	ARFF	ARFF header after this conversion contains dataset parameters (the same set as in GeoDataSet->ValuesArray conversion), @attributes are samples identifiers, gene expression levels are put into @data section
ValuesArray	GeoDataSet	Complete map with dataset parameters is required, data are stored as gene expression levels
ARFF	GeoDataSet	Complete map with dataset parameters is also required, but if ARFF file was created in GeoDataSet->ARFF conversion, whole required data are present in file header

Table 5.3. The description of conversion process between data formats used. The conversion is possible between each of supported data types: text file in *ARFF* format, binary values called *ValuesArray* and *GeoDataSet* object, that is used in microarray experiments. The table also lists additional data not included in Input, but required to produce Output, or excessive data produced, that may be required during reverse conversion process

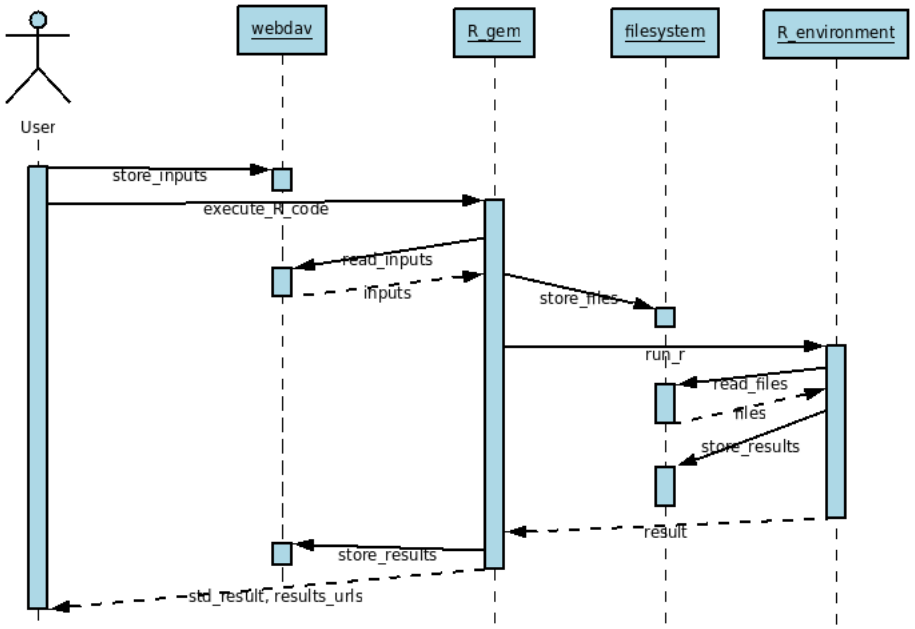


Figure 5.2. R gem running diagram. Data are downloaded from WebDav repository and stored in a local file system. R environment is run in *script* mode. All files created by R are send to WebDav. URLs to these files are returned.

sented in [32]. The knowledge of *R* scripting and Bioconductor packages enables the researcher to perform the statistical analysis in the virtual laboratory.

5.2.2. R Gem

Gem class: `cyfronet.gridspace.gem.r2`

R gem is realized as a binary program wrapper. Scripts written in *R* language are passed to *R* environment. The produced results are obtained from standard output and are read from the working directory. The running diagram is presented in Fig. 5.2. All data needed may be passed directly into the script or put onto a virtual laboratory WebDav repository before executing the gem's method. As parameters the gem acquires script body, that is *code,* and *inputs* map with file names and WebDav paths. The gem creates a temporary directory and then it downloads data from WebDav repository, if passed *inputs* map is not empty. The files are saved in the created directory and the script is passed to *R* environment. The results are read from the standard output. Also temporary directory is examined on new files added. If there is any file created later than those passed as inputs, it is read and put into WebDav repository with an appropriate name. An array with standard output and all the WebDav file names is created as a result of the script execution.

5.3. Data Mining

Data mining is defined as an automatic or semiautomatic process of discovering meaningful patterns in data. Useful patterns allow researchers to make nontrivial predictions on new data. Data mining in the virtual laboratory is available thanks to *Weka* library, which is described in this section.

5.3.1. Weka library

The *Weka* [68] workbench is a collection of state-of-the-art machine learning algorithms and data preprocessing tools. It is written in Java and provides API to use it in own Java applications. Weka also can be used as a separate program. Library includes methods for all the standard data mining problems: regression, classification, clustering, association rule mining, and attribute selection. There are many algorithms available for solving problems in all the categories. Additionally a set of data loaders from different sources is available. Most of machine learning methods can be tuned through properties.

Classifiers Provide learning methods to generate predictions on new instances, also known as supervised learning. The first step is model building on a training set, the second one is testing algorithm on a testing data set, which should be different from the training one. Available in `weka.classifiers.*` packages.

Clusterers Provide algorithms for seeking groups of examples that combine together. Unsupervised learning. Available in `weka.clusterer` package.

Associators Algorithms used for seeking association among features which means discovering any structure in the data that may be "interesting".
Available in `weka.associations` package.

Filters Objects that manipulate data. The whole examples as well as only a single feature may be removed from dataset. Available in `weka.filters.*` packages.

Data loading Loading may be performed from different types of sources - ARFF file, CSV file or directly from a database using JDBC engine.

5.3.2. ARFF file

ARFF file is a default data file type used in *Weka* library. ARFF (Attribute–Relation File Format) is an ASCII text file that describes a list of instances sharing a set of attributes. ARFF files have two distinct sections. The first one is the Header information, which is followed by the Data information. The Header of the ARFF file contains the name of the relation, a list of the attributes (the columns of the data), and their types. Attribute declarations take the form of an ordered sequence of '@attribute' statements. The order in which the attributes are declared indicates the column position in the data section of the file. An attribute has a name and a datatype. *Weka* supports four data types:

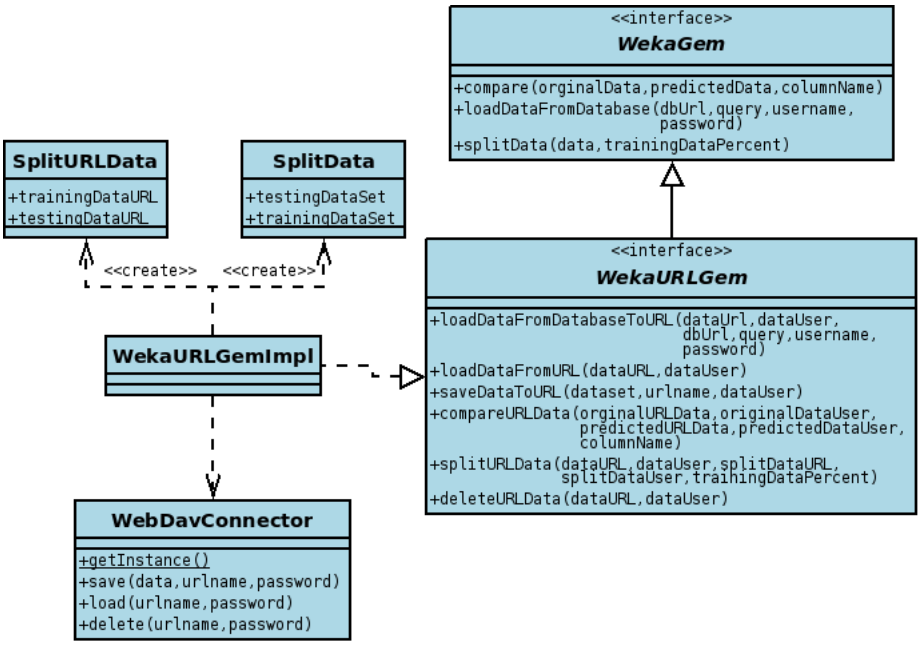


Figure 5.3. Class diagram of Weka data handling part. Basic methods defined in WekaGem are enhanced by the ones that operate on data in WebDav repository.

- numeric - integer, real,
- nominal - list of available values in form: { value1, value2, ... },
- string,
- date.

The data section of the ARFF file contains the data declaration line ('@data') and the actual instances lines. Each instance is represented in a single line, with carriage return denoting the end of the instance description.

5.3.3. Weka gems

Weka library in the virtual laboratory consists of two main parts. Data handling part, presented in Fig. 5.3, is responsible for retrieving data from external database and returning those data in ARFF format, splitting dataset onto training and testing subsets and analyzing classifier prediction accuracy. The second part is composed of Mocca components and depicted in Fig. 5.4. Both parts were developed in Java technology. Each type of machine learning algorithm provided by *Weka* (listed in section 5.3.1) has been packed into a separate Mocca component and can be used to analyze one type of problem: classifying, clustering, associations learning and data filtering. All the components can handle data passed directly as a method parameter or stored in WebDav repository then only data location is passed to the component. The ability of using data stored in central repository may take an effect on the amount of data that is required to pass between Mocca components. Also

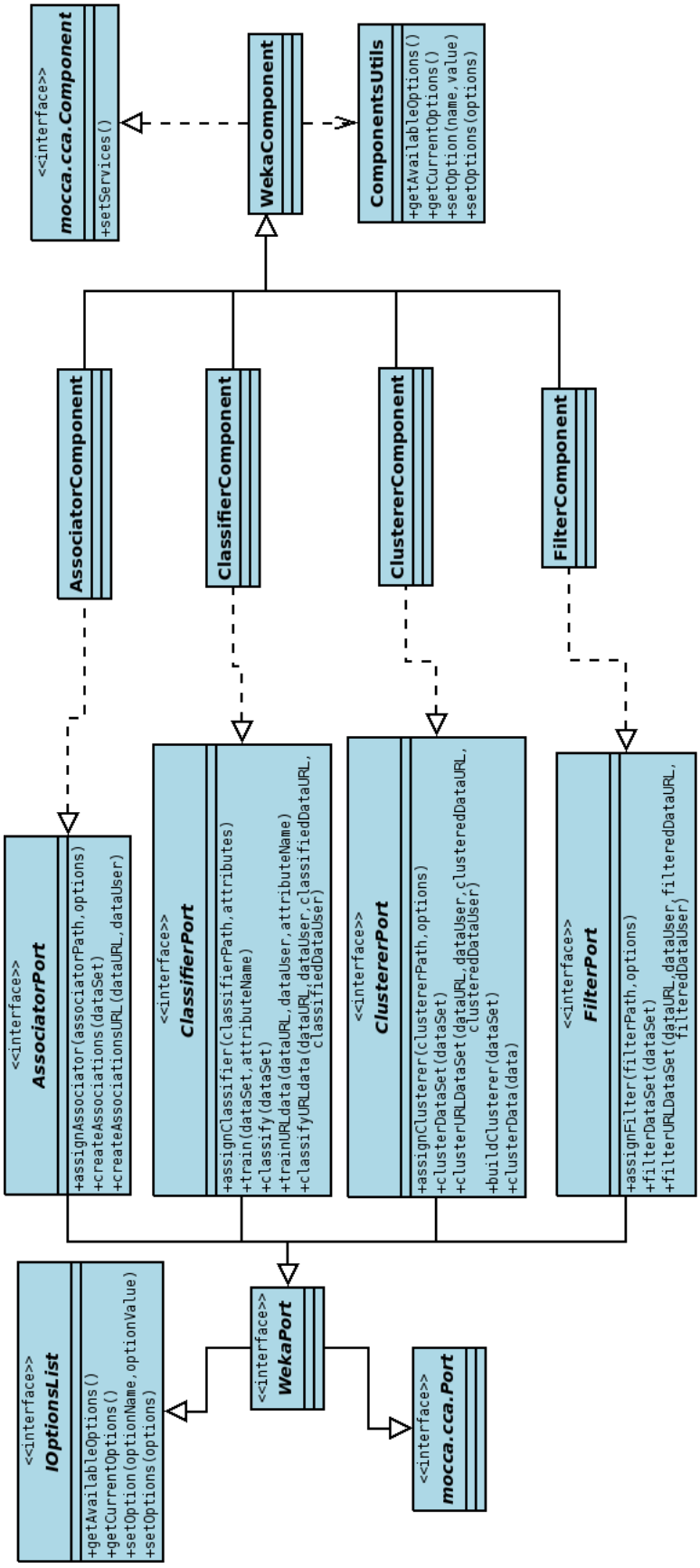


Figure 5.4. Weka Mocca components.

using WebDav repository is a good way to analyze the same data and results in different experiments or with different algorithms.

WekaURLGem

Gem class: `cyfronet.gridspace.gem.weka.WekaURLGem`

Operates on data and allows user to download data from SQL database and return ARFF file. It splits dataset into two parts - testing and training. Classifier prediction can be checked using *compare* method, which tests the value assigned by classifier with the original value presented in the testing dataset. Every method has an equivalent, that operates on the data stored in WebDav repository. Besides, two methods for storing in and loading data from WebDav are available. The gem is created in Web service technology.

Machine learning Mocca components are created in a similar way. Every component requires full Java name of concrete algorithm. It is also possible to provide a set of options that are specific to the selected algorithm. When an algorithm is assigned to a component, calling data mining methods is enabled. Apart from that, every component provides a set of functions to check and change currently set options. Machine learning components are listed below:

WekaAssociator

Gem class: `cyfronet.gridspace.gem.weka.WekaAssociator`

Mocca component which provides methods for learning association rules from the ARFF dataset, that is passed directly or stored in WebDav repository. The found associations are returned as a *String* value.

WekaClassifier

Gem class: `cyfronet.gridspace.gem.weka.WekaClassifier`

Mocca component that provides supervised learning algorithms. When classifier is assigned, the next step is classifier training on a selected attribute. After that new datasets may be classified. Classification creates a new dataset with an attribute column changed accordingly to a classification result.

WekaClusterer

Gem class: `cyfronet.gridspace.gem.weka.WekaClusterer`

Mocca component. Unsupervised learning algorithms are available in *Clusterer* component. Two types of methods are available: the first one returns data clusters description as a *String* value, the second requires two method calls: firstly, clusterer must be built on a dataset, and after that, data passed in *clusterData* method may be assigned to appropriate clusters. An array with assigned clusters numbers' to data instances is created as a result of the second method call.

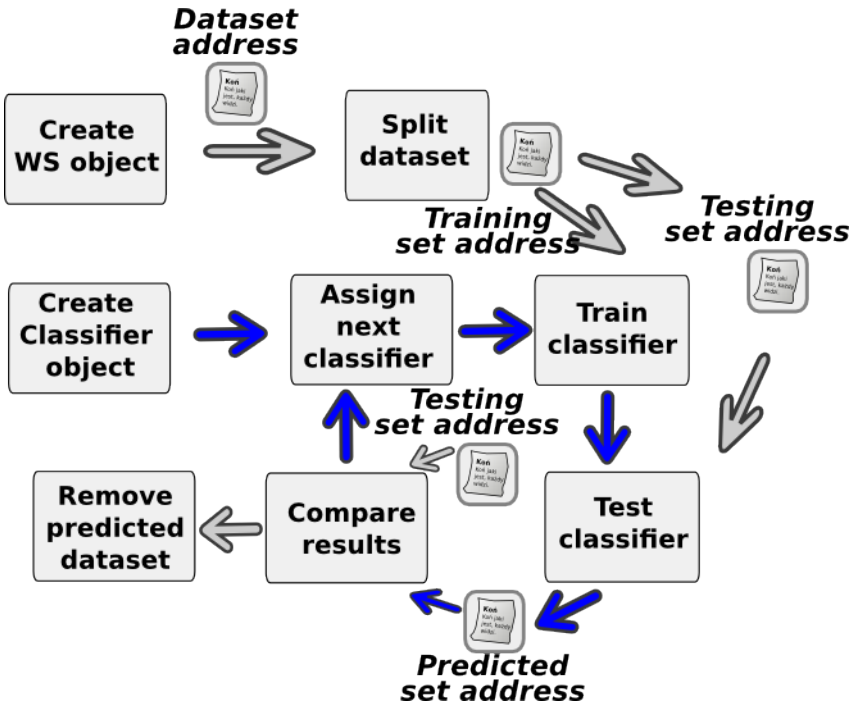


Figure 5.5. Structure of an example experiment for classifiers comparison. Blue arrows are operations on Classifier gem, grey ones are data manipulation performed by WekaURL gem.

WekaFilter

Gem class: `cyfronet.gridspace.gem.weka.WekaFilter`

Mocca component. Within this component the modification of datasets by removing attributes (columns in ARFF file) or instances (data rows in ARFF file) may be performed.

5.3.4. Weka experiments

A sample experiment available in the virtual laboratory contains Weka Classifiers comparison. Ten different classifiers are used in this experiment. Each of them is trained and tested on the same dataset. Prediction quality is printed for each classifier. The experiment has the following steps, as can be seen in Fig. 5.5:

1. Creation of an array with classifiers names;
2. Creation of a Web service proxy object which is used to split dataset into training and testing data, and to delete predicted data;
3. Creation of a classifier object; Splitting dataset into training and testing data. In this example we use dataset Primary Tumor Domain from the University Medical Centre, Institute of Oncology, Ljubljana;
4. For each classifier in the array one should:
   a) Assign next classifier,

   b) Train classifier with the training dataset,
   c) Test classifier and print prediction quality,
   d) Remove predicted data from repository;
5. Remove split (both training and testing) datasets.

5.4. Data Clustering

Clustering algorithms divide data into meaningful groups - clusters - so that intra-cluster similarity is maximized and the inter-cluster similarity is minimized. This step may be used as an initial analysis in data mining. Clustering may also reduce the number of data that should be analyzed in other problems, such as a microarray analysis, when overall data will be reduced only to the selected representatives of the clusters. This section describes clustering libraries and gems made available in the virtual laboratory.

Cluster

Perl module for Cluster 3.0 software has been integrated into the virtual laboratory. This library provides hierarchical clustering algorithm with different distance functions and cluster-distance functions to use, k-means clustering and self organizing maps algorithm. Library was originally developed for clustering gene expression data, but analysis of any data is possible, because the program handles array of double values. For hierarchical clustering the whole dendrogram is returned. If the user wants to get clusters, a cutoff value is required to pass to the method call.

Gem technology: Binary program wrapper.

Cluto

Cluto [2] is a software package for clustering low and high dimensional datasets and also for analyzing the characteristics of various clusters. Cluto provides three different classes of clustering algorithms that operate either directly in the object's feature space or in the object's similarity space. These algorithms are based on the partitional, agglomerative, and graphpartitioning paradigms. Cluto's algorithms have been optimized for operating on very large datasets both in terms of the number of objects as well as the number of dimensions. Cluto provides two types of accepted data formats: graph data, analyzed by *scluster* program and matrix data, analyzed by *vcluster*. The second approach is used in Cluto gem. Matrix data can be annotated with column and row names. For each clustering process different algorithms, similarity functions and criterion functions can be used. Cluto manual contains a complete list of available parameters.

Gem technology: Binary program wrapper.

Weka Clustering

Local Gem that uses Weka Clusterer component to perform clustering. Two algorithms are available: K-Means and Cobweb. Gem creates ARFF file from double data (format converter is used), sets up an appropriate algorithm to Weka Clusterer

component and performs clustering. When results are obtained, another conversion is performed, from ARFF to array of double.

Gem technology: Local Gem, uses WekaClusterer, FormatConverter

Microarray Clustering implementation

It is an own implementation of clustering microarray data library. It allows to use hierarchical agglomerative algorithm or Isodata algorithm. Detailed description is presented in section 8.2.

Gem technology: Implementation

5.5. Reduction of Data Dimension

Principal Component Analysis and Multi Dimensional Scaling are widely used techniques for reducing data dimensionality.

PCA performs a linear mapping of the data to a lower dimensional space in such a way, that the variance of the data in the low-dimensional representation is maximized. In practice, the correlation matrix of the data is constructed and the eigenvectors on this matrix are computed. The eigenvectors that correspond to the largest eigenvalues (the principal components) can now be used to reconstruct a large fraction of the variance of the original data.

Gem technology: Local Gem, uses *R* gem

MDS is a set of related statistical techniques often used in information visualization for exploring similarities or dissimilarities in data. MDS is a special case of ordination. An MDS algorithm starts with a matrix of item–item similarities, then assigns a location to each item in N-dimensional space, where N is specified a priori.

Gem technology: Local Gem, uses *R* gem

Both types of the presented dimensionality reduction may be performed in the virtual laboratory. *R* gem is used to perform these operations, only numeric values (double type) are accepted.

5.6. Summary

In this Chapter we were familiarized with databases that are used in experiments, gems that provide an access to these databases (5 gems) and used data formats. The subsequent sections outlined gems from *"Basic analysis"* layer. Statistical analysis with R (1 gem), Data mining in Weka library (5 gems), Data clustering with one of the available programs (3 gems: Cluster, Cluto, and WekaClustering) and Dimensionality reduction (2 gems) were the main areas that were covered by this layer. Adding these gems required to use various technologies, including binary program wrapping (R gem), library integration using MOCCA component framework, composing gem from the others (ScopDb, WekaClustering, PCA, MDS), as well as own implementation of the algorithms (FormatConverter, GeoDataRetriever, WekaURLGem).

Chapter 6

Protein Sequence and Structure Comparison

This Chapter presents the developed gems and experiments created for performing comparison of proteins that belong to the same family. Firstly, a sequence and structure alignment problems will be described. In the second section a structure of the experiments and the problem solution will be introduced. After that, created gems and classes will be listed. Finally, the results of performed experiments will be depicted.

6.1. Problem Description

Protein structure may be defined at four different levels, as it was presented in Chapter 2.1. Protein comparison is a wide and complex problem. Common structure motifs and conservative areas in a protein family may be found by performing amino acid sequences comparison, secondary structures comparison and 3D structures comparison. Detection of areas in protein that may be responsible for protein function or prediction of ligand binding sites may be done with knowledge about conservative areas. Moreover, sequence and structure comparison performed on all levels of protein description may lead to finding a consensus sequence as well as a consensus structure. Development of protein structure prediction and analysis methods require standardized way to test their quality. Methods that align two crystal structures, one of which may be experimentally determined (e.g. with X-ray crystallography) and the second one originates from in silico experiment, allow to estimate the quality of protein structure prediction. These two types of protein comparison, that are based on the approach presented in Fig. 6.1 are available as

experiments in virtual laboratory. This model is known as *"Early Stage"* and it has been developed by professor Irena Roterman–Konieczna [59, 60] with her team.

Folding simulation path leads from amino acid sequence and contingency table to the complete protein three-dimensional structure. This structure may be compared with that experimentally found and available in PDB database. Protein unfolding path enables comparison of proteins that belong to the same family (or another, freely created protein set). Calculated Φ and Ψ angles values (determining structure between next two amino acids) are changed to the nearest values of the dihedral angles belonging to the ellipse path, by using shortest-distance criterion [21, 19]. The position on this ellipse determines structural code that is assigned to every residue. Protein amino acid sequence may be obtained directly from PDB database or from pdb file. Three types of data: primary structure that comes from amino acid sequence, secondary structure (a sequence of structural codes) and tertiary structure (pdb file with selected chain) combined together create background for similarity search in protein family.

Input data required to perform this experiment is a set of pdb identifiers and selected chains for each protein.

6.1.1. Sequence alignment

Sequence alignment problem may be presented as a compound of two sub-problems: pairwise sequence alignment and multiple sequence alignment[1].

Pairwise sequence alignment

Alignment of two sequences is done with assumption that they are homologous (they share common ancestor). Gaps are inserted into both sequences in order to get the most similarities on the other positions. Two elements are necessary to determine the best sequence alignment:

- Scoring matrix - is used to score each non-gap position in the alignment. They are defined separately for nucleotide sequences and amino acid sequences. Nucleotide scoring matrices are relatively simple - $S(\alpha,\beta) > 0$ if α and β are the same nucleotide, and $S(\alpha,\beta) \leq 0$ if they differ. Designer of scoring matrices for amino acid sequence alignments should take into account several criteria: chemical/physical similarity, observed substitution frequencies, size, charge, etc. Two of the most commonly used models of scoring matrices are PAM and BLOSUM. In the virtual laboratory experiment also an identity matrix is used for aligning structural code sequences.
- Gap penalties function - it may be computed in any form, but most commonly used is affinity gap penalty function, which is defined for l length gap as follows: $W(l) = g_{open}+g_{ext}(l-1)$, where g_{open} is a penalty for creating a gap and g_{ext} is a penalty for extending a gap on next position. Such function definition is related to replications process nature, where 3-nucleotides length gap (for a DNA sequence) is almost as probable as a one nucleotide gap.

[1] Section is based on [35]

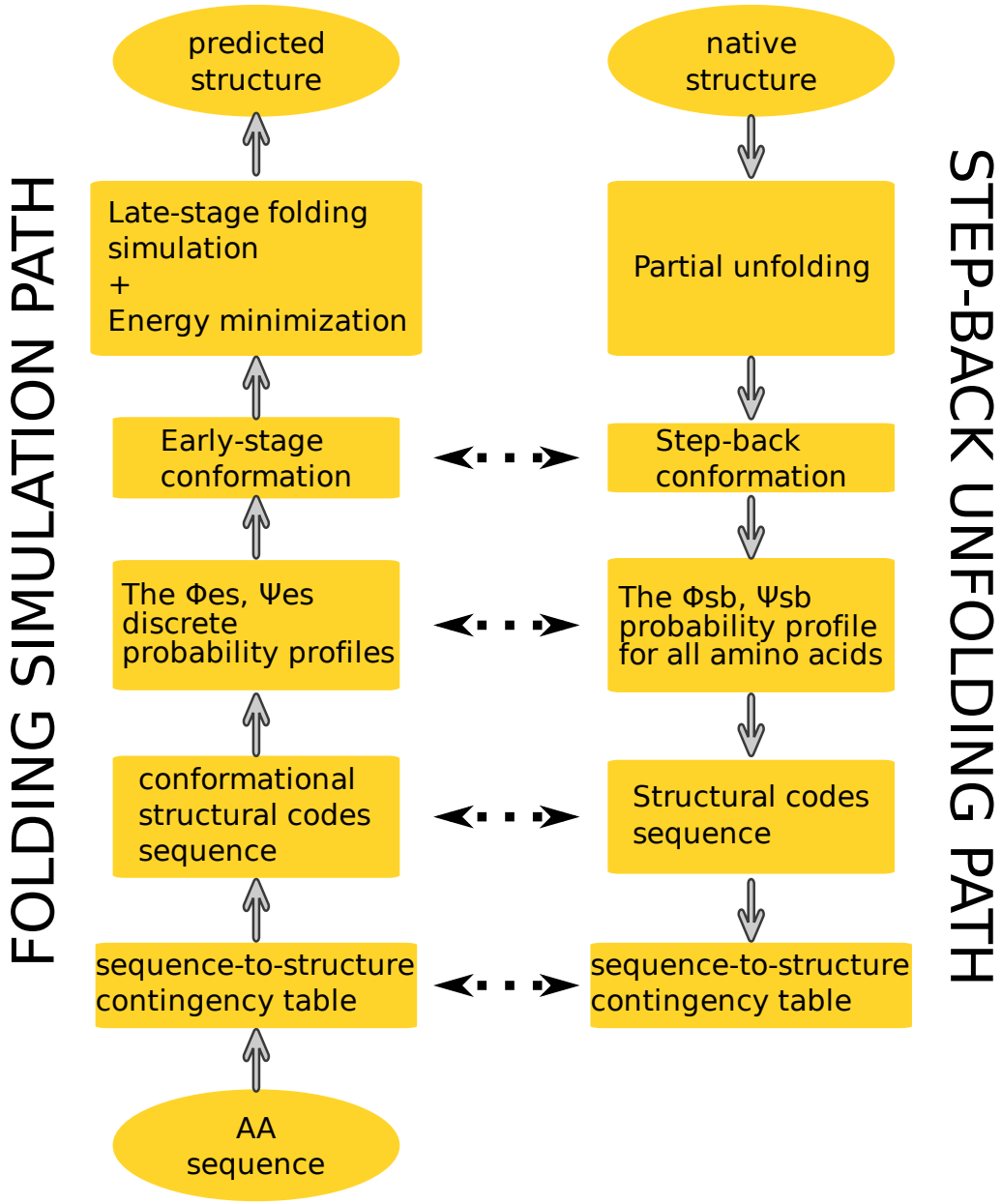


Figure 6.1. Two paths of protein structure analysis scheme. It is based on scheme available in [20]. Dotted horizontal arrows denote equivalent stages of both paths.

Algorithms for solving the sequence alignment problem are often designed using dynamic programming technique. It is a method of breaking a problem apart into reasonably sized sub-problems, and using these partial results to compute the final answer. When sequences are compared in their entirety, a global alignment is performed. Sometimes it is required to match only a small fragment of the sequence. Local alignment is a technique where the best matching of two subsequences is found and areas that are not similar are not taken into consideration (score for alignment in those areas is not counted and do not produce abysmal value).

The alignment score for the same pair of sequences may be slightly different if we use another gap penalties function or scoring matrix, but similar, strongly conservative areas will be matched irrespectively of the selected algorithm parameters.

Multiple sequence alignment

Time complexity for the multiple sequence alignment problem (finding global optimum for n sequences is strongly dependent on the sequence length and is an NP-complete problem) causes that using direct methods is impracticable. The most widely used approach to multiple sequence alignments uses a heuristic search known as a progressive technique. All progressive alignment methods require two stages: a first stage in which phylogenetic tree, called a guide tree, that represents relationships between the sequences is constructed, and a second step in which the multiple sequence alignment is built by adding the sequences sequentially to the growing MSA according to the sequence distances defined in the guide tree. Progressive alignments cannot be globally optimal. A variety of algorithm improvements, like an evolution distance usage in scoring matrix and modifying of gap penalties along to gap sequence position have been developed.

One of the most popular multiple sequence alignment solution has been implemented in ClustalX software (also available as Web service under ClustalW name). Another programs that solve this problem are Muscle, T-Coffee, KAlign.

6.1.2. Structure alignment

Structural alignment[2] attempts to establish equivalences between polymer structures based on their shape and three-dimensional conformation. The outputs of a structural alignment are a superposition of the atomic coordinate sets and a minimal root mean square deviation (RMSD) between the structures. The RMSD of two aligned structures indicates their divergence from one another. Most of structure alignment programs are able to compute amino acid sequence alignment which is based on created protein atoms superposition. This type of protein comparison is particularly important when proteins with low sequence similarity are compared and evolutionary similarity cannot be found with using standard sequence alignment methods. Reducing computational complexity is done by analyzing only a selected subset of all atoms in each protein. This approach is based on physical/chemical atom properties. For simplicity and efficiency, often only the alpha carbon positions

[2] http://en.wikipedia.org/wiki/Structural_alignment

are considered, since the peptide bond has a minimally variant planar conformation. There are plenty of available implementations of different algorithms, that solve structure alignment problem. Some of them are able to compare only two structures, another may be used in multiple structure alignment problems. The most popular are: Distance Alignment Matrix (DALI), Combinatorial extension (CE), GANSTA+, MAtching Molecular Models Obtained from THeory (MAMMOTH), Rapid Alignment of Proteins In terms of DOmains (RAPIDO), Sequential Structure Alignment Program (SSAP).

6.2. Experiment

The implemented experiment follows the step–back path of protein comparison, which was presented in Fig. 6.1. The experiment diagram is presented in Fig. 6.2.

The input data are protein pdb identifiers with selected chains for each protein. Researcher may put it directly into the script or pdb identifiers may be obtained from SCOP database if the researcher uses protein family name. In the second case all chains for each protein in the family are considered as input data.

For each chain and protein in the dataset the required information is created. This experiment section is organized as follows (used gem or classes names are presented in brackets):

- Protein crystal structure defined in pdb file is obtained (*DbFetch*),
- Pdb file is filtered for only those parts that define atom positions that belong to the selected chain (*PdbUtils*),
- Amino acid sequence is created for residues described in pdb file (*PdbUtils*). It is possible to download a complete sequence from PDB database, but there are small differences between sequences obtained directly from the database and residues described in pdb file for some proteins. The sequence obtained from pdb file is often shorter than sequence from PDB database. Because of the fact that pdb file is used in comparison with other two levels, the first approach to create amino acid sequence is used,
- Structural codes generation, based on pdb file (*EarlyFolding*),
- Crystal structure data filtering for structure alignment step (*PdbUtils*). Mammoth gem is used in the experiment to perform structural alignment step. This program uses only alpha carbon atoms positions in computations. In order to reduce the amount of data passing to the gem, the positions of other atoms are removed from pdb file. This step is optional.

All data types created in this experiment section are stored in the local file system. Those data are rather constant, it means that the determined sequence, the secondary and the tertiary structure for selected chain in the protein wouldn't be different if we want to compare the same protein with another one during next experiment run. This step makes experiment performing faster, especially if next experiment run differs only in small subset of input data. Before each step of this section the script tries to find specified file in the file system. If this file is present

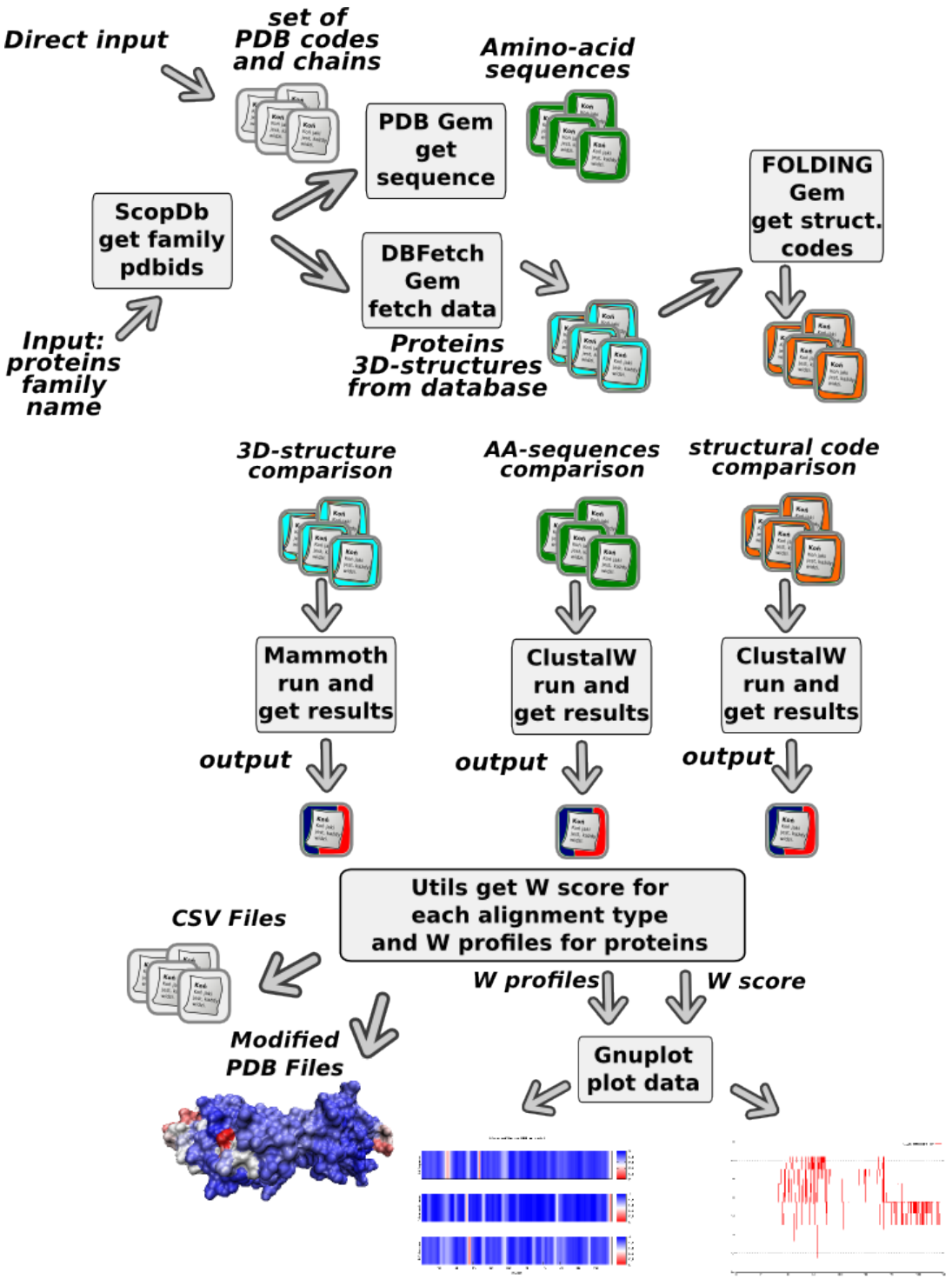


Figure 6.2. Scenario of the Protein Sequence and Structure Comparison experiment. For input pdb codes the amino acid (AA) sequence and pdb files are downloaded, then structural codes (SC) are computed. AA and SC sequences are aligned in ClustalW, while crystal structures are aligned in Mammoth. For aligned sequences W score and W profiles are computed. Finally, resulted plots and modified pdb files are generated.

```
>15C8_L
DIVLTQSPAIMSASLGERVTMTCTASSSVSSSNLHWYQQKPGSSPKLWIYSTSNLASGVPAR
>1AD9_L
DIQMTQSPSTLSASVGDRVTITCRSSKSLLHSNGDTFLYWFQQKPGKAPKLLMYRMSNLASG
>1ADQ_L
YVLTQPPSVSVAPGQTARITCGGNNIGSKSVHWYQQKPGQAPVLVVYDDSDRPPGIPERFSG
>1AE6_L
DIVMTQAAPSVPVTPGESLSISCRSSKSLLHSNGDTFLYWFLQRPGQSPQLLIYRMSNLASG
>1CLY_L
LMTQIPVSLPVSLGDQASISCRSSQIIVHNNGNTYLEWYLQKPGQSPQLLIYKVSNRFSGVP
>1CLZ_L
DVLMTQIPVSLPVSLGDQASISCRSSQIIVHNNGNTYLEWYLQKPGQSPQLLIYKVSNRFSG
>1DQD_L
DIVLSQSPAIMSASPGEKVTITCSASSSVSYMHWFQQKPGTSPKLCIYTTSNLASGVPARFS
>1EMT_L
DIQMTQTTSSLSASLGDRVTFSCSASQDISNYLNWYQQKPDGTIKLLIYYTSSLRSGVPSRF
>1HZH_L
EIVLTQSPGTLSLSPGERATFSCRSSHSIRSRRVAWYQHKPGQAPRLVIHGVSNRASGISDR
```

Figure 6.3. Data passed to ClustalW service in FASTA format. Lines starting with '>' are data identifiers, the remaining lines are input sequences.

information is read from the disk, if not - data downloading and appropriate step is performed.

Analyzing three levels of protein description is the third part of the experiment. Amino acid sequence alignment as well as structural code sequence alignment is performed with *ClustalW* gem. For performing computations on ClustalW Web service data must be structured in one of the accepted formats (an example of used FASTA format is presented in Fig. 6.3). Also options list is necessary to run computations. Options that have been set in the experiment are: 'email', '*outorder*' and '*matrix*'. Option '*outorder*' is set to ensure that the aligned sequences will be returned in the same order as input sequences were passed to the service, '*matrix*' option is a selection of scoring matrix: for amino acid sequences the *'blosum'* matrix is chosen, while identity matrix *'id'* is used during structural codes alignment. Computations in the *Clustal* service are asynchronous, *jobid* is assigned to each request. *ClustalW* provides methods for checking job status for the task with a known *jobid*. Waiting for finishing computation and checking status is done in a loop. When a job status is 'DONE' the alignment results may be retrieved from the service; this data requires encoding from *Base64* format. The best sequence alignment found is returned as a result (example presented in Fig. 6.4). Crystal structure alignment is performed with *Mammoth* gem usage. As input data the program requires a set of pdb files to analyze and information on which chain and in which the protein computation should be performed. In this experiment data passed to *Mammoth* are preprocessed during the data gathering part, in this step pdb files are filtered (non–used chains are removed, as well as the positions of atoms other than alpha carbon). *Mammoth* program creates sequence alignment based on protein atoms conformation. The output of *Mammoth* gem is standardized to the *ClustalW* output obtained during sequence and structural codes alignment.

The aligned sequences are next analyzed in *ClustalWUtils* for finding W score [20]. W value is computed for every position in alignment sequence as follows:
$W = \log_{10}\left(\frac{F}{(N/M)+1}\right)$, where F is a maximal code frequency for a particular position,

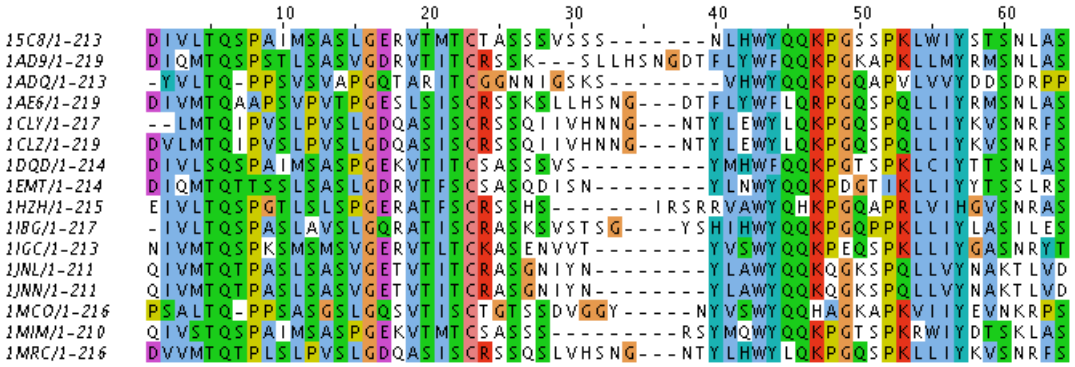


Figure 6.4. Example of sequences alignment for protein family. Visualization is done in Jalview [67]program.

N - is the total number of aligned sequences and M denotes the number of codes ($M = 8$ for structural code alignment and $M = 21$ for amino acid sequences, indels marked as '–' are also included). The resulted W value is then averaged, in this experiment averaging window has a length of five residues, but can be easily changed by a researcher. W value is then normalized to $[0; 1]$ range, in order to compare the results obtained in experiment runs, when protein families with different number of members are analyzed. W value counted as it was depicted before, together with protein sequence alignment, allow to count W profile for every protein in the input dataset. From the aligned sequence only residue / structural code positions are selected (insertions/deletions are omitted) and appropriate W value is added to the profile. This approach is presented in Fig. 6.6.

W values for all three types of an alignment (amino acid sequences, structural code sequences and crystal structures) are then plotted as a linear chart with *Gnuplot* gem. Gnuplot script with one data series on a line chart is defined in *script.gpl* file (that is why three charts are created). W profiles for every protein as well as chain in input dataset are also created with *Gnuplot*. Plotting script is defined in *multplot.plt* file. This script is parametrized with a protein name, a selected chain, a chain length (the number of residues) for each protein before that script is passed to *Gnuplot* gem. With that script multichart plot is created - three W profiles for every protein are drawn as a Blue - White - Red maps. Plotting W scores for sequence alignments on one chart is not possible because every alignment may create a resulting aligned sequence with a different length. W profiles for one protein have always the same length (number of amino acid codes, structural codes and amino acid codes based on crystal structure alignment is constant).

The last stage in the experiment is creating output data - CSV files for every protein and chain. Those files contain all W profiles for appropriate protein and chain assigned to residues. CSV file is organized as follows: First line is the header, in the next lines in the first column the residue number is put, then successively the amino acid W profile, the structural code W profile and the crystal structure W profile. The same data are put into pdb files. Because the pdb format is restricted [13],

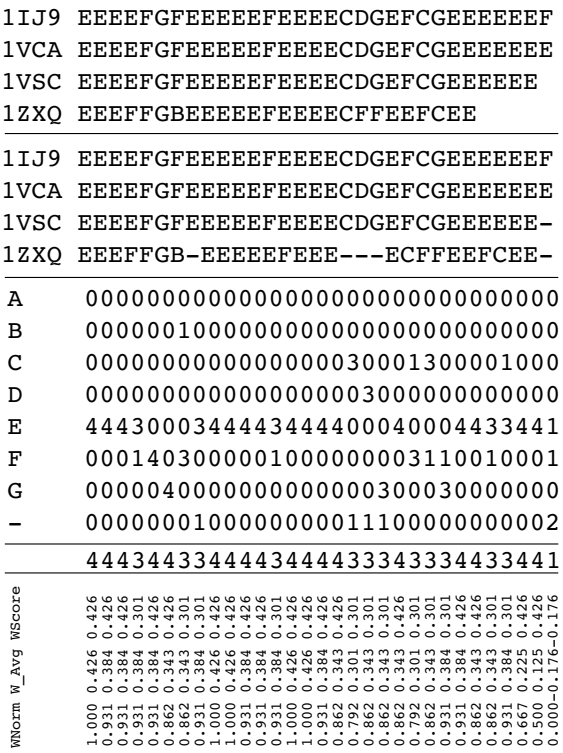


Figure 6.5. W score computing method. Figure presents: input sequences (structural codes), aligned sequences, codes frequency on every position, highest frequency on particular position, W scores (basic W formula used, averaged with 3 codes averaging window and normalized). This example is based on the one presented in [20].

Figure 6.6. W protein profile computing example. This Figure presents W profile for 1ZXQ protein and structural code sequence.

based on the original pdb file, two new ones are created. File with *'_aa_sc'* suffix added to base name has amino acid W profile put into *'temperature'* field and W profile for structural codes alignment in *'occupancy'* field. The second file, with *'_td'* suffix contains W profile values of crystal structure put in *'temperature'* field. Modified pdb files allow researchers to check W profiles values in external protein visualizers. In many of them protein structure colouring based on *'temperature'* (e.g. Jmol, VMD - in this program *'temperature'* field is called *'beta'*) and *'occupancy'* (e.g. VMD, Jmol doesn't support this field) is available. The experiment finishes with creating html file, that collects links to all charts created during experiment execution. The complete code of the developed experiment is listed in Appendix B.1.

6.3. Gems and Classes Used in Experiment

In the previous section the detailed experiment structure was presented. In this section gems and classes used in experiment and those available as alternatives are listed. Gems are divided into groups by responsibilities.

6.3.1. Data gathering

The first experiment stage is gathering of required data. Those data are pdb codes list and chains, amino acid sequences, structural codes sequences and protein crystal structures. Gems that allow researchers to accomplish this step are listed below.

ScopDb

Gem class: `cyfronet.gridspace.gem.bioinfo.data.ScopDb`

Gem provides methods to access to the Structural Classification of Proteins database [53]. It retrieves pdb identifiers that belong to the protein family with name passed as a method parameter. In order to find the pdb code, at first protein family identifier is searched on the appropriate HTML page from RCSB portal. With this identifier a new query in a XML format is created and passed to *Pdb* gem. Resulted pdb codes are returned in String array. Using these data in the experiment requires additional knowledge about chain names for each protein. This knowledge may be obtained from PDB database by using appropriate method on *Pdb* gem.

Technology: Local Gem, uses methods provided by *Pdb* gem.

Pdb

Gem class: `org.pdb.Pdb`

Pdb gem is an external Web service that provides methods for querying Protein Data Bank database. Four methods, selected from those available in complete Web service

API[3], are available in the virtual laboratory. Those methods perform: retrieving amino acid sequence for selected *structure* and *chain ID*, retrieving list of available chains in structure, checking chain length and querying PDB database with complex queries in a XML format.

Technology: External Web service. Provided by RCSB [9, 16].

DbFetch

Gem class: `uk.ac.ebi.Dbfetch`

Gem provides an access to various biological databases. Every database has unified identifier and can handle specific query. Queries are build in "DB_identifier:query" format. During execution of protein sequence and structure comparison experiment this gem is used to obtain protein crystal structures which are available as pdb files.

Technology: External Web service. Provided by EBI [42].

EarlyFolding

Gem class: `cyfronet.gridspace.gem.EarlyFolding`

The main problem that this gem solves is protein structure prediction. It also allows to analyse protein structure and computes structural codes for the examined crystal structure (Step-Back path presented in Fig. 6.1). This method is used in the experiment.

Technology: External Web service. Provided by CM UJ.

PdbUtils

Additional class created for this experiment. Provides and realizes interface that allows user to: select only atom records from pdb file that belong to selected chain; select only Cα atoms; or create amino acid sequence that is based on residues described in pdb file for selected chain. The two additional methods for putting W score into pdb files for appropriate amino acid residues are available.

Technology: JRuby class.

6.3.2. Sequence alignment

In the virtual laboratory sequence alignment is performed with the usage of programs provided as Web services by European Bioinformatics Institute [42].

ClustalW

Gem class: `uk.ac.ebi.ClustalW`

[3] Complete API description is available at
http://www.rcsb.org/pdb/software/static.do?p=/software/webservices/PdbWebService.html

ClustalW [66] is a fully automatic program for global multiple alignment of DNA and protein sequences. The alignment is progressive and considers the sequence redundancy. Trees can also be calculated from multiple alignments.

Technology: External Web service

ClustalW2

Gem class: `uk.ac.ebi.ClustalW2`

ClustalW2 [44] is a second version of ClustalW application.

Technology: External Web service

Muscle

Gem class: `uk.ac.ebi.Muscle`

MUSCLE [28, 29] stands for MUltiple Sequence Comparison by Log-Expectation. MUSCLE is claimed to achieve both better average accuracy and better speed than ClustalW or T-Coffee, depending on the chosen options. Multiple alignments of protein sequences are important in many applications, including phylogenetic tree estimation, secondary structure prediction and critical residue identification.

Technology: External Web service

TCoffee

Gem class: `uk.ac.ebi.TCoffee`

T-Coffee [54] is a multiple sequence alignment program. The main characteristic of T-Coffee is that it will allow to combine results obtained with several alignment methods. For instance if there is available an alignment coming from ClustalW, an other alignment coming from Dialign, and a structural alignment with some users modification, T-Coffee combines all that information and produces a new multiple sequence having the best agreement with all these methods. By default, T-Coffee will compare all sequences two by two, producing a global alignment and a series of local alignments (using lalign). The program will then combine all these alignments into a multiple alignment.

Technology: External Web service

6.3.3. Structures alignment

The second main task in protein sequence and structure comparison experiment is performing a multiple alignment of crystal structures. There are three gems that are able to perform multiple structure alignment (*Mammoth*, *MultiProt*, *SSM*) and one that may be used to pairwise structure alignment (*Dali*). Pairwise alignment is an useful technique to analyze protein structure prediction quality, when *in silico* created crystal structure is compared to that experimentally found, available in PDB database. In the virtual laboratory *EarlyFolding* gem performs a protein structure

prediction. All mentioned gems require as input data pdb codes mapped to protein structures. Aligned amino acid sequences are created as results of all analyses using crystal structure alignment.

Mammoth

Gem class: `cyfronet.gridspace.gem.structure_comp.Mammoth`

Mammoth gem is a wrapper for *Mammoth-mult* [48] application. *Mmult* program is a Backbone Atom Alignment algorithm implementation [56]. From whole pdb file, on which Mammoth program operates, only alpha carbon (Cα) atoms positions are taken into consideration. All input structures are placed in one pdb file, where every protein structure is terminated with TER line. As a result of program execution a new file set is created. From those files the most important is the one with *'aln'* extension. Multiple sequence alignment based on structure conformation is written in this file. Mammoth program renames input proteins to its own identifiers. The gem has to remember the order in which analyzed sequences were placed in input file and it has to perform inverse renaming after *mmult* execution. Mammoth-mult is written in FORTRAN language. This language does not allow to dynamic array creation, so all parameters, such as the number of protein that may be aligned at once, have to be defined before compilation. The default constraint to number of protein structures was set to 30. *Mmult* used in Mammoth gem has been recompiled, maximal number of aligned proteins is set to 50. There is also a constraint on the protein chain length - chains with more than 880 residues are not allowed.

The diagram of Mammoth gem running is presented in Fig. 6.7. Performing structure alignment request is handled as follows:

1. Create temporary directory for handling particular request;
2. Create *mmult* input, all structures passed to the gem as a *pdb_identifier => crystal structure map* are written to one input file, every two structures are separated by TER line, only Cα atoms position are put into the file;
3. Execute *mmult* program;
4. When *mmult* is finished, read aligned sequences from *'aln'* file. Other created files may be also retrieved but it must be stated in the options passed to the gem;
5. Remove temporary directory;
6. Modify *'aln'* output - replace *mmult* protein identifiers with those present in input data. Identifiers are remembered during *mmult* input file creation. Modified *'aln'* output is a result of the method execution.

One of the most important *mmult* features is its quickness. For maximal allowed problem size the program finishes in several minutes.

Technology: Web service, binary program wrapper

MultiProt

Gem class: `cyfronet.gridspace.gem.structure_comp.MultiProt`

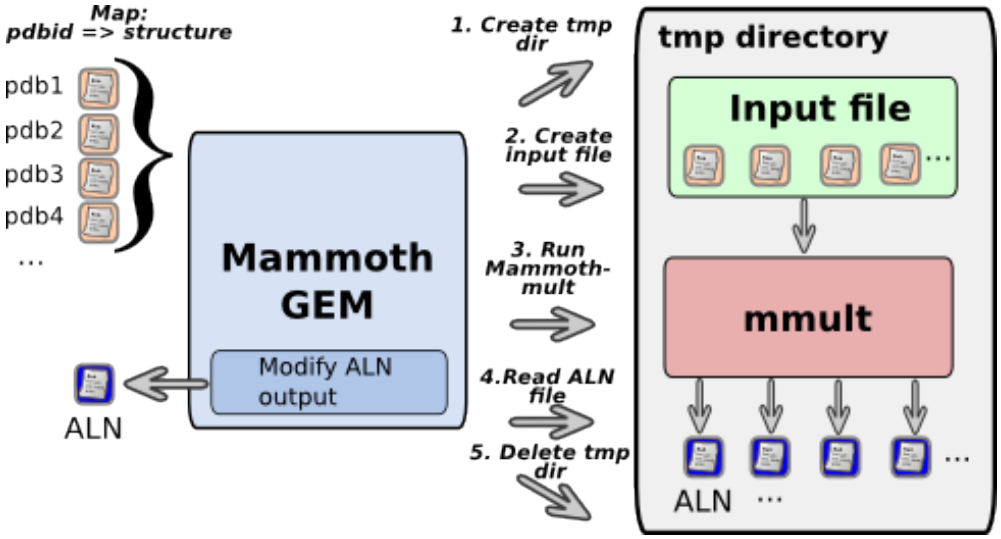


Figure 6.7. Mammoth gem running diagram. Input pdb files are stored in a temporary directory as a one file which is the input for Mammot-mult. Aligned sequences are read from 'aln' file and returned after identifiers modification.

MultiProt [64] is an acronym for Multiple Alignment of Protein Structures. Computations performed in *MultiProt* program are based on a complete protein geometry. This approach is more time consuming than Mammoth-mult algorithm, especially for large protein sets. *MultiProt* does not have strictly defined memory limits for number of aligning structures. *MultiProt* software is distributed as a binary program for Unix systems, so that some problems may occur if it is run on non standard architectures. The gem uses several programs belonging to *MultiProt* and *Staccato* software packages. In order to obtain the sequence alignment four of them are used. *MultiProt* gem running diagram is depicted in Fig. 6.8. During analysis process following steps are performed:

1. Create temporary directory for handling particular request, a second directory named *trans*, for storing partial results is created inside;
2. Store input structures passed in a map (the same type as in Mammoth gem) as files in the local file system, MultiProt uses separate files for every structure;
3. Run *multiprot* with all files passed as the command line parameters (wildcard *.pdb is used);
4. When *multiprot* is finished, select from the results an appropriate file. The files have names that match to pattern /NR_res.sol/, where NR is a number of aligned structures. File with the greatest number of aligned structures, which should be the same number as the size of input data set, is selected as input to the next step;
5. Run *trans_mult* with selected /NR_res.sol/ file, a number of the best solution available in this file (aligned marked as 0 solution) and the name of inner temporary directory;
6. Inside trans directory run *mkdssps* program;

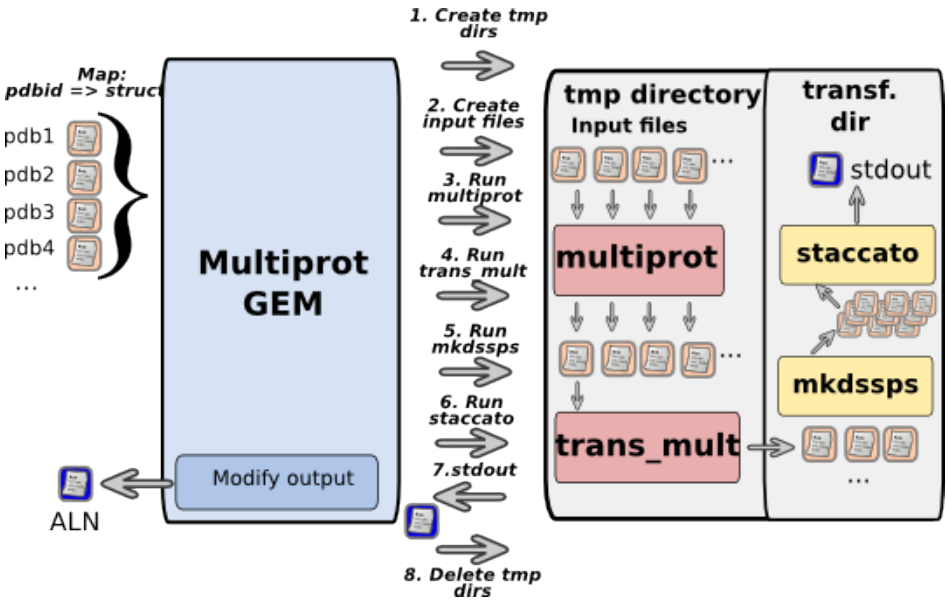


Figure 6.8. MultiProt gem running diagram. Four programs are used to create sequence alignment on the basis of structure alignment: *multiprot*, *trans_mult*, *mkdssps* and *staccato*. Results are read from standard output.

7. Run staccato program, read standard output where aligned sequences can be found;
8. Remove temporary directories;
9. Modify output, remove unnecessary information and change staccato identifiers to the same as was passed in input map.

Performing *MultiProt* computations takes remarkably longer than *Mammoth*. For the same dataset, that consists of 45 proteins that belong to the IGG family, running time is near 75 minutes and it is about 6 times longer than the same computations performed in *Mammoth* gem.

Technology: Web service, binary program wrapper

SSM

Gem class: `cyfronet.gridspace.gem.structure_comp.Ssm`

SSM is a wrapper for *Secondary Structure Matching*[4] service provided by EBI. SSM gem communicates with the service in HTTP protocol. Each request is handled in a separated thread. For computation request for pdb files following steps are performed:

1. Create temporary directory for handling particular request, store input structures passed in a map;
2. Create compressed tarball from input pdb files;

[4] http://www.ebi.ac.uk/msd-srv/ssm/

3. Prepare POST request with created tarball as an argument, send it to the SSM server. The Session ID is read from returned HTML page.
4. Remove temporary directory
5. Wait for results.
6. Download results in FASTA format, return this file as a result

This service does not accept any options to set.

Dali

Gem class: `uk.ac.ebi.Dali`

Dalilite [36] is a tool for a pairwise structure comparison. It may be used for comparing protein structure prediction quality, performed in *EarlyFolding*, with original crystal structure, obtained from the PDB database by using *DbFetch* gem.

Technology: External Web service, provided by EBI [42]

6.3.4. Results preparing and analysis

The last part of the experiment is a numerical analysis of obtained sequences and structures alignments and results visualization. The gems and objects used in this part are described below. Additionally, PdbUtils class described in section 6.3.1, is used to update pdb files with W score profile values.

ClustalWUtils

Gem class: `cyfronet.gridspace.gem.structure_comp.ClustalWUtils`

Gems created for W score computing, algorithm is described in section 6.2. Beside the methods for W score and W profile computing, the gem provides a method that analyzes aligned sequences strings. Most of programs return sequences divided into some parts. This method joins appropriate parts of aligned strings and returns a map with protein identifiers as keys and complete sequences as values.

Technology: Web service, own implementation

CSVUtils

This class provides one method for creating CSV files with W profile values for selected protein. The file is organized as follows: protein name and selected chain are written in the first line, column headers are put into second line and then next rows contain W profile values for appropriate residues (residue number in first column), amino acid W profile in column 2, structural code W profile in column 3 and crystal structure W profile in column 4.

Technology: JRuby class, own implementation

GnuPlotUtils

A simple class for transforming data into format that Gnuplot accepts. Two methods are available - the first one for creating data used in plain charts with one

data series (x value is residue number, and y is W score) and the second for creating charts as color maps (x is in [1..2], y is a residue number and z is W score).

Technology: JRuby class, own implementation

Gnuplot

The gem is a wrapper for a binary program with the same name. Its detailed description is presented in Chapter 9. Two Gnuplot scripts for drawing charts are prepared. The first script defines linear chart with W score for aligned sequences. Three separate charts of this type are created in the experiment - one for amino acid sequence alignment score, one for structural codes score and the last one for crystal structures alignment score. The script is accessible as a *script.gpl* file. The second script creates multiplot chart. On this plot three profiles are drawn as a color map plots (*pm3d* technique in Gnuplot). W profile values are represented in Red - White - Blue color scale. Script is defined in *multiplot.plt* file. This plot type is used for creating plots for every protein and every chain from among input data.

Additional scripts

A set of functions used for storing and reading data generated during experiment execution (e.g. pdb files with protein crystal structure, structural codes for protein) is additionally defined in *results_handling.rb* script file. There are also available some functions for storing results, like plots, CSV files, updated PDB files defined in this script.

6.4. Experiment Run and Results

The prepared set of gems and scripts is suited to solve a protein comparison problem. The key parts of an experiment may be computed with gems used interchangeably. These key parts are data retrieval (gems that are available to use are: *ScopDb*, *DbFetch*, *Pdb* and *EarlyFolding*), sequence alignment (many services from EBI available: *ClustalW*, *ClustalW2*, *Muscle*, *T-Coffee*) and structure alignment (mostly *Mammoth* and *MultiProt* gems are used, also *SSM* gem is available). The defined experiment structure and the input data organization allow researcher to easily change the initial data set and compare results obtained with using different gems.

The experiment was executed to analyse the mechanism of signal transduction in immunoglobulins. The proteins belonging to the immunoglobulin super-family like ICAM, VCAM and IgG were analyzed. As a processor of sequences and structural codes alignment, the *ClustalW* gem was used. The *Mammoth* gem was employed for computing structures alignment. Proteins and their chains selected from used super-families to analysis are listed below:

- ICAM: 1ZXQ:A
- VCAM: 1VSC:A; 1IJ9:A; 1VCA:A
- IgG: 1CLY:L; 1CLZ:L; 1EMT:L; 2DD8:L; 2IG2:L; 1MCO:L; 2JB6:L,A; 2ZPK:L,M; 2ADF:L; 1ADQ:L; 1AD9:L; 2J6E:L; 1N0X:L; 1HZH:L; 1NSN:L; 1MIM:L; 1AE6:L;

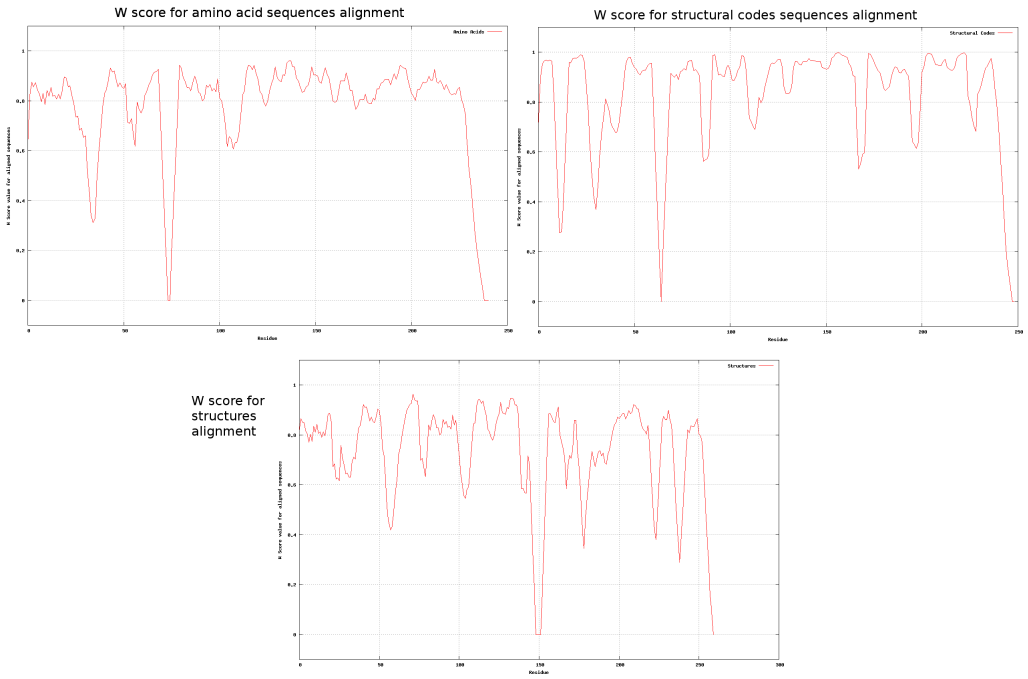


Figure 6.9. Experiment results. Three types of W score value for ICAM + VCAM + IgG super-family are presented: amino acid alignment (upper-left), structural code alignment (upper-right) and structures alignment.

15C8:L; 2VXV:L; 1IBG:L; 1JNL:L; 1JNN:L; 1MRC:L; 2MCG:1,2; 1MRD:L; 1MRE:L; 1MRF:L; 2AGJ:L; 2JB5:L; 1UZ8:L; 1DQD:L; 2Z4Q:A,B; 1IGC:L; 2CMR:L; 2VXQ:L; 4BJL:A,B

An organization of the input data as a hashmap with *protein id* as a key and *array of used chains* as a value together with Ruby language features allow researcher to easily combine data belonging to different input data groups. An experiment was performed separately for proteins belonging to each super-family (except ICAM) and for all combinations of listed super-families (ICAM + VCAM, ICAM + IgG, VCAM + IgG and ICAM + VCAM + IgG). The results for W score obtained for ALL group (ICAM + VCAM + IgG super-family) are presented in Fig. 6.9. Additionally, W profiles for protein 2DD8 and chain L are depicted in Fig. 6.10, where all levels of comparison have been included. A Red-White-Blue color scale has been used to show protein's conservation regions. A dark blue color means that selected region is highly conservative. A protein structure visualization was performed in VMD software. As a result of the experiment execution a set of W profiles pictures for each protein, CSV files with numerical data about W profiles and PDB files that can be visualized are produced.

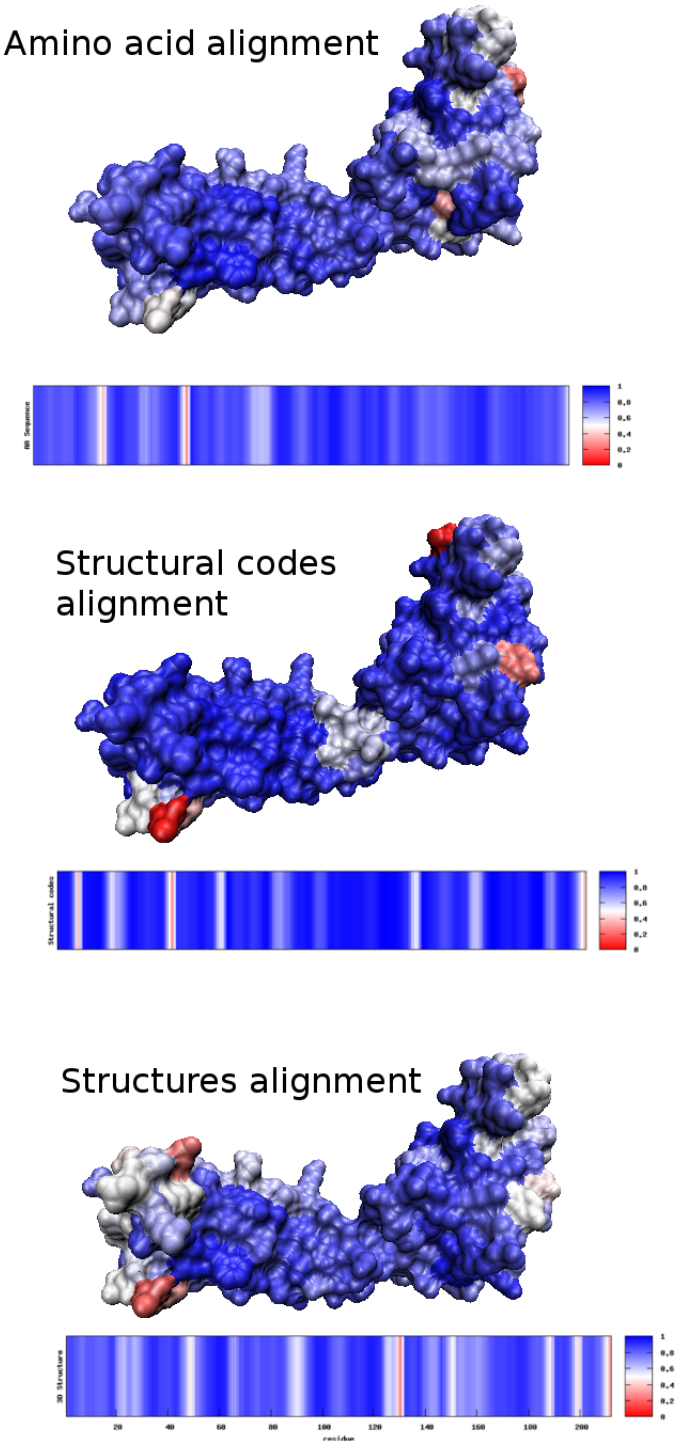


Figure 6.10. Results of Protein Sequence and Structure Comparison experiment. W profiles for 2DD8 protein are shown. Protein structures are visualized in VMD, while color maps are generated during experiment execution in Gnuplot.

6.5. Summary

In this Chapter the experiment created for solving protein structure and sequence comparison problem was presented. Thirteen new gems and some additional classes used in the experiment were developed in order to prepare complete solution. The created gems are applicable in the following problems: data retrieving (4 gems), multiple sequence alignment (4 gems), multiple structure alignment (4 gems), W score computing (1 gem). Three additional classes were also developed: the one for input data handling and two others for results handling and analyzing. This experiment was executed with using various protein families as an input dataset: ICAM (one protein), VCAM (three proteins), IgG (36 proteins, 41 selected chains), and all their combinations. Obtained results were analyzed and will be described in *"Conservative structural element in proteins engaged in immunological signal transduction"* paper [61], which is currently in preparation.

Chapter 7

Comparison of Services for Predicting Ligand Binding Site

The comparison of services for predicting ligand binding site on a protein surface is the core of this Chapter. In the first section the problem will be stated. Later, every service will be presented. Afterwards a specific usage of the general task queuing architecture, introduced in section 4.3.1, is described. Finally, the created experiment is shown.

7.1. Problem Description

One[1] of the greatest challenges for researchers in bioinformatics field is understanding of the functioning of biological systems. Structural genomics, as the most comprehensive approach, is used. The aim of structural genomics is determination of the primary and tertiary structures of all proteins of a given organism, identification of function and evaluation of functional integrity of these proteins. Another goal, justifying the huge investments already made in structural genomics initiatives, is the ability to predict druggability of a particular protein based solely on its 3D structure. Three different groups of prediction of ligand binding sites strategies can be distinguished. Methods that are tailored to detect pockets and clefts on the basis of pure geometric criteria constitute the first group. CastP [27], PocketFinder (an implementation of LIGSITE [34]), Ligsite_csc [37] and Pass [18] services, that belong to this group, have been made available in the virtual laboratory. The prediction methods of the second group, in addition to structural data use biophysical and/or chemical properties. That includes computations of pKa, electrostatic energy, sol-

[1] Section is based on [57]

vent mapping, physical potential, favourable regions for van der Waals (CH_3) probes on the protein surface or hydrophobicity deficiency. From this group, Q-SiteFinder [45] and Fod [22] have been made available in the virtual laboratory. The third group of methods relies on knowledge derived from biochemical data and different types of databases. Services that implement this type of algorithm are ConSurf [43], SuMo [40, 39] and WebFeature [46]. Services from every group require protein's 3D structure, in the form of pdb file. As a result of performing computations in any of these services, a list of chains and residues that are found as binding site is created.

In most cases mentioned services are available as WWW portals. Integration of these portals into the virtual laboratory require wrapping a communication with a service in HTTP protocol. The integration performed in the scope of this thesis allows users to run large scale analyses that are executed for more than one protein on many services.

7.2. Description of Available Services

This section describes every gem available in the virtual laboratory, that may be used to perform ligand binding site prediction.

CastP

Computed Atlas of surface topography of proteins [17, 27]. CastP belongs to the first group of services that performs prediction on the basis of pure geometric criteria.

Availability http://sts.bioengr.uic.edu/castp/
Methods pdb file, pdb id
Results pockets atoms (.poc), list of residues that form the biggest pocket (.txt)
Technology Web service, HTTP communication wrapper

ConSurf

Server for the Identification of Functional Regions in Proteins [43] belongs to the third group of methods that relies on knowledge derived from biochemical data and different types of databases (in this case SWISS-Prot and UniProt sequence databases are used). Only one chain from passed protein structure may be analyzed at once. In order to perform computation for the whole protein the request must be repeated with different value of *'chain'* parameter.

Availability http://consurf.tau.ac.il/
Methods pdb file, pdb id. Analysis based on pdbid is a searching of precalculated results available in the ConSurf database.
Results Amino Acid Conservation Scores, Confidence Intervals and Conservation Colors (.gradesPE) and list of residues with Conservation Color equal or greater than specified (.txt)
Technology Web service, HTTP communication wrapper

Fod

FOD [22] is a representative of the second group of algorithms.

Methods pdb file
Results $\Delta\tilde{H}$ profile (.dat), list of residues with $\Delta\tilde{H}$ higher or equal to maximum $\Delta\tilde{H}$ minus quarter of the $\Delta\tilde{H}$ range
Technology Web service, binary program wrapper

Ligsite_csc

Ligsite_csc [37] is an implementation of the LIGSITE algorithm [34].

Availability http://gopubmed2.biotec.tu-dresden.de/pocket
Methods pdb file, pdb id
Results list of atoms (ligand.txt), list of pockets (pocket.txt), list of residues (.txt)
Technology Web service, HTTP communication wrapper

Pass

PASS [18] service belongs to the first group of methods, that predicts binding sites on the basis of the geometric criteria.

Methods pdb file
Results probe spheres (probes.pdb) and list of residues forming the biggest pocket (.txt)
Technology Web service, binary program wrapper

PocketFinder

This is one of two services provided by Faculty of Biological Sciences in University of Leeds. It belongs to the first group of methods. This service has limitation on protein size. Protein cannot have more than 10 000 atoms.

Availability http://www.modelling.leeds.ac.uk/pocketfinder/
Methods pdb file, pdb id
Results atoms forming pockets (.pdb), list of residues for the largest pocket (.txt)
Technology Web service, HTTP communication wrapper

QSiteFinder

QSiteFinder is the second service provided by Faculty of Biological Sciences in University of Leeds [45]. This service has the same limitation on protein size as *PocketFinder*.

Availability http://www.modelling.leeds.ac.uk/qsitefinder/
Methods pdb file, pdb id
Results atoms forming pockets (.pdb), list of residues for the largest pocket (.txt)
Technology Web service, HTTP communication wrapper

SuMo

SuMo [39] service belongs to the third group of methods [40]. Calculation are much longer than in other services - for a protein of medium size it takes several minutes. Available service resources are also limited - often calculation request is put into a queue where it is waiting (even some hours) for start of processing.

Availability http://sumo-pbil.ibcp.fr/cgi-bin/sumo-welcome
Methods pdb file, pdb id
Results text file (txt), list of residues for the highest ranked hit(s) (txt)
Technology Web service, HTTP communication wrapper

WebFeature

WebFeature [46] service belongs to the third group of methods. Unfortunately, running more than one task at the same time is not allowed.

Availability http://feature.stanford.edu/webfeature/
Methods pdb file, pdb id
Results archive of points files (sig.tar) and list of residues reflecting one of specified z-score specificity cutoffs: 100%, 99% and 95%(.txt)
Technology Web service, HTTP communication wrapper

ResultsConverter

Every service returns results that are written in a specific format. In some of those formats the numbers of residues, that are found as binding site are put directly into the file. Others require additional data and computation in order to assign appropriate residues to the found binding site. Comparison of the results from different services requires common data format. Therefore a set of format converters, that are available under unified interface, as ResultsConverter gem, has been created. Every converter provide set of methods that are responsible for:

- `get_atoms_from_pocket` - looks for residues that are assigned to the largest pocket irrespective of chain in which residues are placed, this method returns correct values only for structures that contain only one chain
- `get_atoms_and_chains` - returns a list of residues that are present in the largest pocket, and additionally chain identifiers assigned to every residue, are given
- `get_atoms_for_chain` - this method allows user to search the largest pocket in which at least one residue belong to chain that identifier was passed as method parameter

ResultsConverter gem is a Facade (design pattern, [31]) to specialized converters. Converter is selected on the basis of its name, provided as a parameter to the method. Conversion rules from a service specific format to the common format and short description of these formats are presented in Table 7.1.

Service	File(s) type	Description and rules of conversion
CastP	'poc'	File contains information about residues, that are found as pockets, these lines are rewritten from original pdb file, each line however contains a number (rows 67-70) identifying a pocket number. The pocket with the highest number is assigned to the most probable binding site. This file is parsed to two-level map with pocket number as a primary key, chain identifier as a secondary key and residues numbers as values. Pocket with the highest number or that with appropriate chain is selected from the map, it depends on method that is used.
ConSurf	'gradesPE'	File contains information about residue conservation (color value: 1 - variable, 9 - conservative). A supply of conservative level is required to extract residues that are found as binding site. Residue number, which is present in line identifier column, are added to list of binding site residues if a residue color is grater or equal to the supplied value.
Fod	'fod'	Every residue has assigned value that characterizes its ability to form binding site. Threshold value is set to $thr = max - 0.25 * (max - min)$, all residues that have score equal or greater than a threshold are added to the list of returned residues.
Ligsite_csc	'pocket', 'pdb', 'ligand'	'pocket' file defines representers of found pockets. Pocket radius for which computation was performed is required as a method parameter. As the next step, from 'ligand' file all numbers of residues that belong to any of found pocket are read. Residue is assigned to pocket if it is close enough to pocket center, residue position is read from 'pdb' file.
Pass	'probes', 'pdb'	'probes' file defines probes position, which are clustered with hierarchical clustering algorithm and 'cut_off' value set to 2.0. To every cluster residues from 'pdb' file are ascribed. Assignment is based on residue position and its distance to the probes that form the cluster. As a result the largest cluster is returned.
PocketFinder, QSiteFinder	'pdb'	'pdb' file contains added sections with atoms positions that belongs to found binding sites. Those positions are compared with ATOMs lines and from those lines residue numbers are read.
SuMo	'text'	Results are grouped into sections that are ordered from the highest prediction score. Residues belonging to every section are directly included into section description. The highest ranked section contains resulted residues numbers. More than one section may have the same score, in this case results from all sections are joined
WebFeature	'significant hits'	Files set from extracted tar archive and file with threshold values for every model are required to perform analysis. Additionally z-score specificity cutoffs is required as method parameter. Every line in file contains a residue identifier and z-score value. If z-score is not less than parameter residue is added to result.

Table 7.1. Results format description and rules of conversion from every specific format to common format for ligand binding site prediction services.

7.3. Integration of Gems Using Task Queuing System

Every gem described in this Chapter that was created and integrated into the virtual laboratory is based on the general task queuing system architecture that was described in section 4.3.1, if it is a web-based portal wrapper, or on the wrapper architecture, depicted in section 4.3.2, for gems that have to us specified program locally. Some of the classes were specialized for solving ligand binding site prediction problem. System architecture is depicted in Fig. 7.1.

Classes that are mentioned below were specialized during system development.

`ProteinService` defines gem interface and queuing system parameters. Available methods for every gem (except general methods listed in Service class) are:
`get_results_from_file` - analysis is based on provided pdb file
`get_results_from_pdb` - analysis is based on provided pdb structure identifier

`ProteinTask` defines task description. Two static methods for creating and setting up *Task* object are added. Additionally, enumerable *ProteinTaskType* class is added. Enum type defines if task is created for pdb file or for pdb identifier request.

`ProteinTaskManager` two new methods for creating concrete task type are added to the general *TaskManager* class.

`ProteinTaskProcessor` this object has the knowledge about Task type and it calls ProteinTaskAnalyzer method depending on it.

`ProteinTaskAnalyzer` is a task information analyzer, derived classes are specialized for appropriate web-based service. All classes share common interface.

The task creation process is performed when computation request is sent to the gem. The sequence diagram presented in Fig. 7.2 describes task creation process for pdb file computation request. This process is performed as follows:

1. Send request from the Experiment script to the selected Service.
2. If the list of files that are requested to download and the selected options are correctly defined, request is passed to *Manager*. The appropriate request parameters are created.
3. *Manager* calls static method from *ProteinTask* class in order to create *Task* object. The options passed as request parameter are then set to *Task* object. Writting file, file name and task identifier to *Task* object is also done in this step, performed inside *ProteinTask's* static method.
4. The created object is returned to *Manager*.
5. The newly created *Task* is added at the end of FIFO queue, where all tasks that not have been computed yet are waiting.
6. The method tries to run the first waiting task from FIFO queue. Running new task is depicted in Fig. 7.3. Running new tasks performed when new task is added to queue and when any of actually performed computations is finished ensures that task starvation is not possible [65].

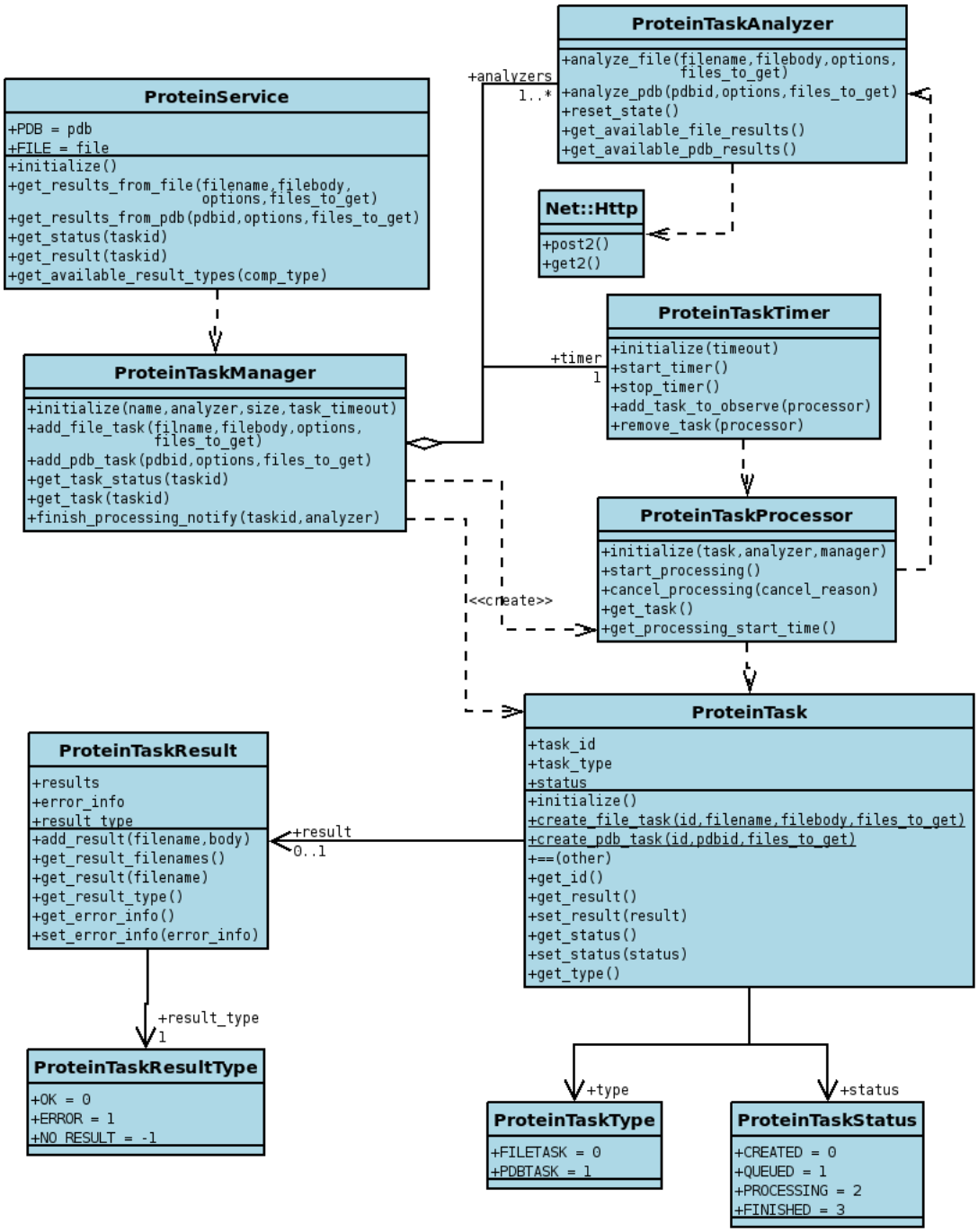


Figure 7.1. Class diagram for binding site prediction services. It is a concrete realization of the general task queuing architecture, presented in section 4.3.1. *ProteinService* class defines the system interface. *ProteinTaskAnalyzer* is responsible for communication with WWW portals. Additionally, *ProteinTask*, *ProteinTaskManager* and *ProteinTaskProcessor* are specialized version of the general ones from task queuing system.

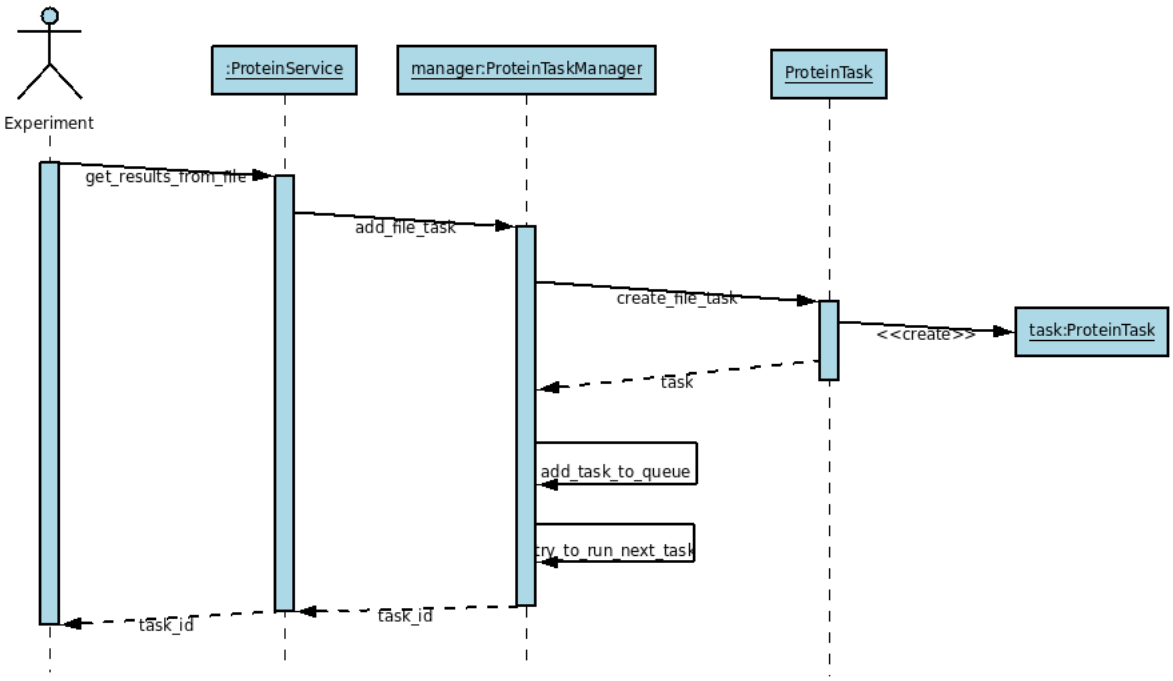


Figure 7.2. Sequence diagram for Task creation process performed when `get_results_from_file` method is called.

7. Created task identifier is returned to the Service and then to the Experiment script.

Performing task computations is the next important process. Its sequence diagram is shown in Fig. 7.3. This process is organized as follows:

1. The notification about the necessity of running a new task to analyze is passed to the task manager. There are two types of notification: first, new task to analysis from *ProteinService* call, and second, information from processor about finish currently processing task.
2. The *try_to_run_next_task* method, which is responsible for handling tasks running, is called.
3. The first analyzer object from the available analyzers queue is dequeued. If the queue is empty, it means that all the analyzers are currently processing assigned tasks, running new task is finished and the task stays in *QUEUED* status.
4. New processor object is created for gathered task and analyzer objects. The processor runs analysis in a new thread.
5. The newly created *Processor* is added to the *Timer's* observed processors list. This is done for securing processing time limit and ensuring that no one task will be starved.
6. Analysis execution on *Processor* is started. *PROCESSING* status is set on *Task*.
7. Task analysis process is executed in a separated thread. New *ProteinTaskResult* object is created as a result of executing analysis method on *Analyzer* object.

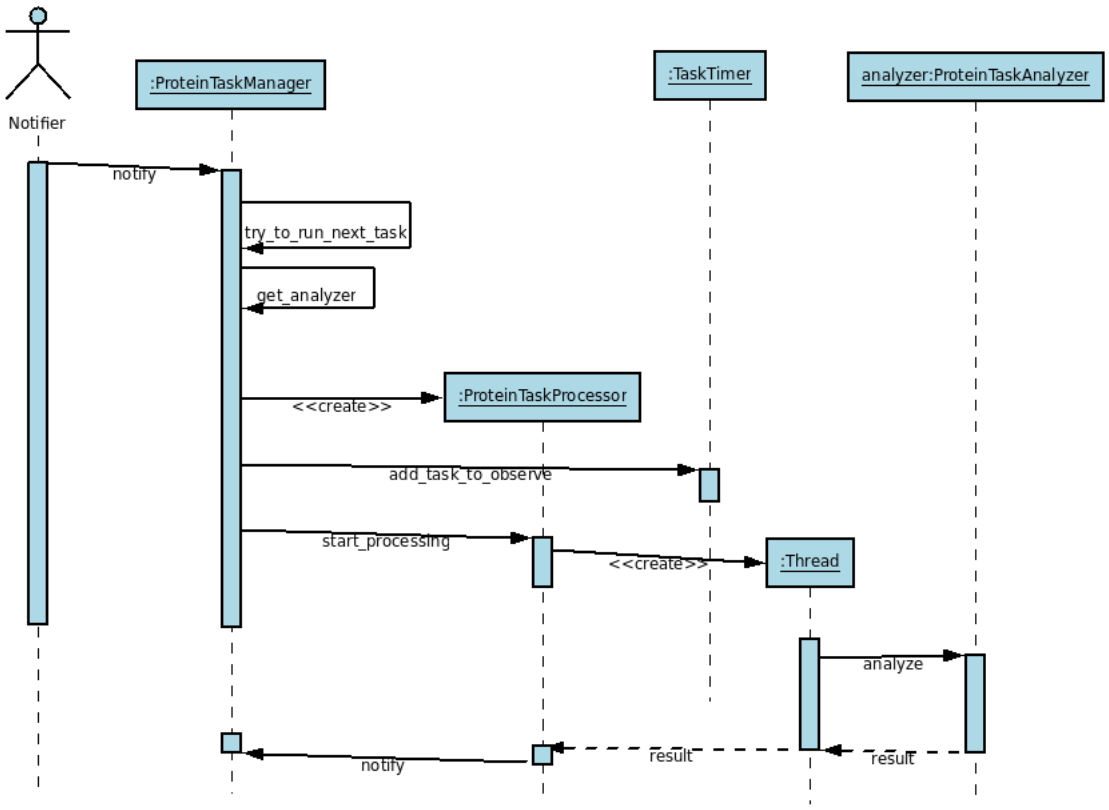


Figure 7.3. Sequence diagram for running Task. Manager creates a new ProteinTaskProcessor to which a reference to an Analyzer is added. Processor runs analysis in new Thread. Timer observes analysis and terminates it if maximal analysis time is exceeded.

8. *Processor* adds result obtained from *Analyzer* to task and performs finished analysis notification on *Manager*. *Manager* adds *Analyzer* to available analyzers queue, then *FINISHED* status is set on *Task* object, which is added to processed tasks list. If there is any task waiting in the tasks queue, *Manager* tries to run next *Task* (process starts again from point 2).

7.3.1. Analyzers

There are two types of ligand binding site prediction services available in the virtual laboratory. The first group are web-based applications, and the second group are locally available, binary applications. Respectively, two groups of objects responsible for analysis are created. Firstly, objects that use HTTP protocol for communication and parsing HTML responses, and secondly, analyzers that are local binary program wrappers. Class diagram for analyzers' part of the system is depicted in Fig. 7.4. *ProteinTaskAnalyzer* is a main class and it is a base class for specialized analyzers objects. This class provides basic methods used by analyzers from web-based services group. From all methods we can list: POST request headers defining and preparing, options handling, validations and adding options to headers,

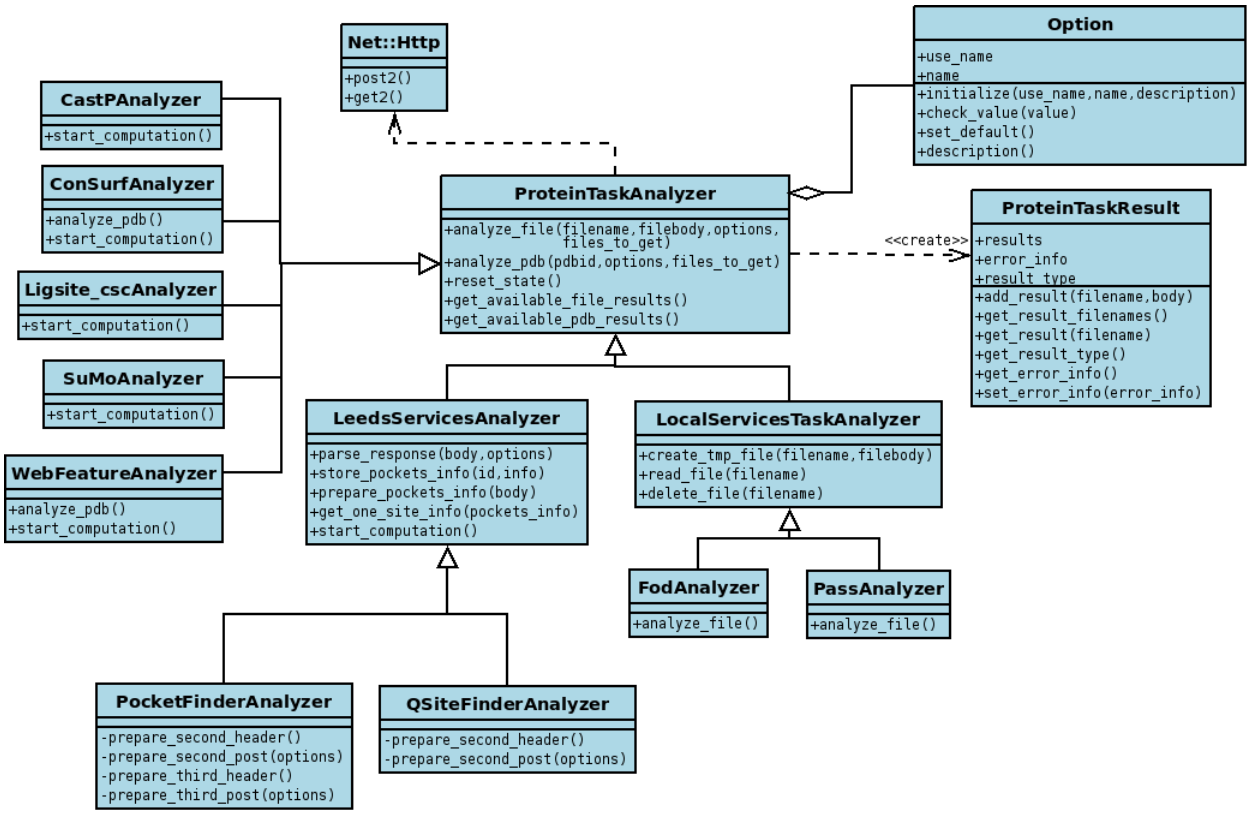


Figure 7.4. Class diagram for analyzers part of the system.

checking HTTP response code and downloading files. Every specialized analyzer redefines *start_computation* method. This method is responsible for performing complete communication process with appropriate WWW service. Detailed communication process with every WWW service was developed on basis of web pages content, and HTTP packets analysis, performed with using *Wireshark* software. Analyzers from WWW services group use a standard Ruby module, *Net::Http*, for communication, sending requests and receiving responses from services. The second group, analyzers that execute binary programs, have been developed in a way described in section 4.3.2. *FodAnalyzer* and *PassAnalyzer* belong to this group. Both of them derive from *LocalServiceTaskAnalyzer* class. This class is responsible for creating temporary data files, reading and removing those files.

7.3.2. Options validation

Almost every one of the services allows users to tune the analysis process by setting selected parameters. A set of classes for defining available options and their value ranges was developed in order to securing correct communication with binding site prediction services. Those classes allow also to checking values assigned to options that user has set. Class diagram for option mechanism is presented in Fig. 7.5. *Option* class constructor creates object and assigns name, usage name

and option description to it. Usage name is a string value that is meaningful for developers, it is an accepted name that may be set in the parameters map, passed to a gem in an analysis request. Options descriptions are used when a user calls the method on the service that lists available options, acceptable values and their descriptions. The most important method, that *Option* object provides, is checking correctness of option value, if it has been set in acceptable option values range. This method is called *check_value*. *Analyzer* object defines options list and sets up their value ranges, when it is created. Those options are put into a map, which is indexed with *use_name* values of created options. When the analysis request is called, passed options connected with this request are checked. This process is performed as follows: firstly, the option object is got from options map, based on option name passed in parameters. After that, on the selected object, *check_value* method is called, with the value passed in the input parameters map as a method parameter. When an option has not been found (because wrong name was set as a key in the input parameters map) or option's value is not correct (a value for a selected key does not belong to the acceptable values range), analysis process is not performed. Error information is returned to user, as a result of this failed analysis.

Three specialized options classes are available to use. These classes differ by accepted value types. These classes are:

StringOption String analysis is based on the defined regular expression. Value is checked if it matches the expression.

NumericOption NumericOption analysis is available for double type. Minimal and maximal values, by default set to 0.0, create acceptable values range. Checking option's value is done by comparing a passed value to option's minimal and maximal attributes. If the value is greater or equal than the minimum and less or equal than the maximum it is accepted. Redefined *description* method adds to the standard description an information about accepted range of values.

SelectOption Option value may be selected only from a set of defined values. Redefined *description* method adds to the standard description a list of acceptable values.

Every *Option* sub-class redefines the *check_value* method. This method returns true if a value has been accepted and false in other case.

7.3.3. The parameters of the created services

Available services have been parametrized with different configurations. The configurable parameters set consists of maximum number of parallely processed tasks, processing timeouts and service name. Values set in every service are listed in Table 7.2.

7.3.4. Running services

There are two additional Shell scripts created in order to conveniently run and stop all the gems created for this experiment. A '*servers_start.sh*' script runs all

Gem name	Max parallel tasks processing	Task processing timeout [s]	Default files to retrieve
CastP	10	360	'poc', 'poc_info'
ConSurf	5	3600	'gradesPE', 'blast', 'alignment'
Fod	5	3600	'fod'
Ligsite_csc	5	3600	'pdb', 'pocket', 'ligand'
Pass	5	3600	'pass'
PocketFinder	2	3600	'pdb', 'pockets'
QSiteFinder	2	3600	'pdb', 'pockets'
SuMo	10	86400	'text'
WebFeature	1	3600	'significant_hits'

Table 7.2. Parameters values for available binding site prediction services.

Ruby Web services. A '*servers_shutoff.sh*' script firstly tries to find a PID value for each service and then sends a KILL signal to an appropriate process.

7.4. The Prediction Experiment of Ligand Binding Sites

The experiment that performs comparison of available services on a dataset created from locally stored pdb files has been developed in the virtual laboratory. This experiment may be executed from *ViroLabEPE* environment. All input data files have to be placed in one directory. The path to this directory is set as a variable value in the main experiment script. A set of used services is also configurable and may be changed by the researcher. A set of tasks to be computed on every service is created for a list of pdb files that are present in input directory and for a set of used services. An initial subset of complete task list is created for every used gem. A size of initial tasks subset is the same for every service. This value is not connected with gem's maximal number of parallely processing tasks parameter. A subset size value is configurable in experiment script. Described experiment model has been developed in order to optimize data sending time. It is important especially for large data sets. The experiment scenario is depicted in Fig. 7.6.As a result of sending a task to a gem, a task identifier is returned. When the initial dataset sending is completed, the experiment enters into a loop. In this loop the checking status of every task from every service is performed. If a task has status *TaskStatus::FINISHED*, it is taken from the gem and a new computation request is sent to this gem, if the task queue for this service, created in the experiment, is not empty. The task result is examined on its status if an analysis has finished with success or error. If the task result type is an error, it may be caused by problems in communication with WWW services, a not correctly defined PDB file, web-based services limitations,

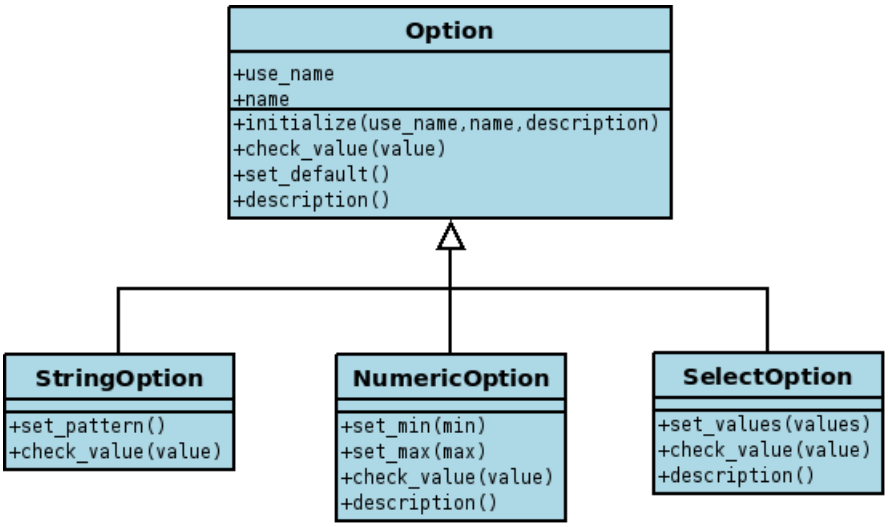


Figure 7.5. Class diagram for Options mechanism.

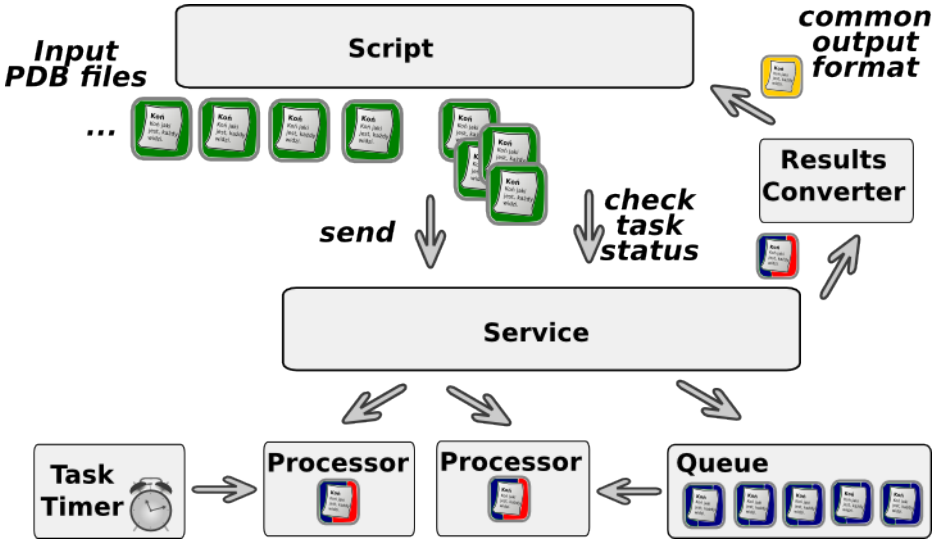


Figure 7.6. A model of the experiment of comparison of the binding site prediction services. Tasks created from pdb files are sent to the service which is created as a task queuing system. Results of analysis from each service are converted to the common format.

e.g. on analyzed protein size, or by exceeded time of an analysis. As a result of an analysis, one file with an '_ERROR' suffix in its name and an error information included in its body is written to the disk. If task's result type is success (OK), all the files retrieved from a service are written to the disk. From these files a subset of required files for further analysis is selected. This subset is dependent on a used service, and it is described in Table 7.1, in section 7.2. Selected files are passed to the *ResultsConverter* gem in order to convert specific file formats to a simple form of an information about predicted ligand binding site. A common result format consists of only a chain name and residues numbers that form a predicted binding site. For every analyzed protein, defined in pdb file, Jmol visualizing script is created, when results from all services are available on the disk. This script allows users to select and color residues predicted as a binding site by the specified service. It is done by providing a service name. When any of services returns an error for a pdb file, this service is omitted in a script and a group of residues for this service cannot be selected in the visualization software. This part of the experiment is presented in Fig. 7.7.The complete code of the experiment is listed in Appendix B.2.

7.4.1. Additional classes and scripts

A set of classes and script files is created besides the main experiment's script and gems depicted in section 7.2. Additional classes and scripts that provide methods for managing tasks and analysis results are listed below.

file_utils.rb

A script. Provides functions to access to files: files reading and writting, generating file name from a service name, a pdb file name, a chain and a result type, and also method for reading pdb files from disk and putting these files into map.

service_utils.rb

A script. This script includes functions for aiding tasks sending to services. These functions are: examining pdb file on protein chains; checking available computation type for selected service, if it must be done separately for every chain (*ConSurf* service) or if it may be done for complete protein; definitions of required files to download from every service lists; definition of files extensions required to store files into the disk; unpacking *WebFeature* archives and defining map with data (a file name as a key and a file content as a value).

TaskListCreator

A class. It provides method for creating list of tasks left to be computed. This list is created from an input data directory content, a list of used services and a content of the directory with results. It is useful especially when the experiment performing has been interrupted. By using this object the experiment may be continued from near the same state as it was before interruption. Additionally a report about state may be generated.

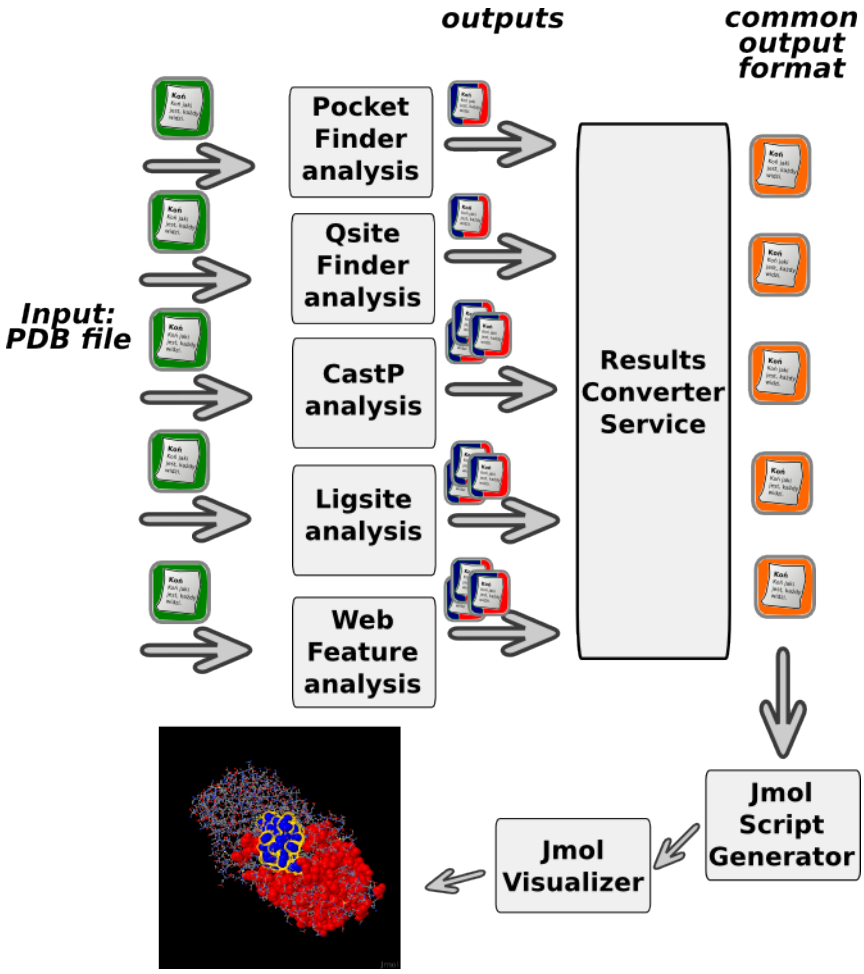


Figure 7.7. Results conversion and visualization in the experiment of binding sites prediction services comparison. Common format of each result is analyzed and put into the Jmol visualization script.

task_utils.rb

A script. This script provides functions that operate on tasks maps. These functions are: getting tasks left to be computed by the service; getting next task to be computed by the service; removing the task from the not computed tasks list; adding and removing the task from a currently processing tasks list.

Those classes and script allow developers to design well structuralized main experiment script.

7.5. Summary and Results

The seven external, web-based services for predicting a ligand binding site in protein were integrated as a gems in the virtual laboratory with usage of the developed task queuing mechanism (See section 4.3.1). Additionally, two binary applications

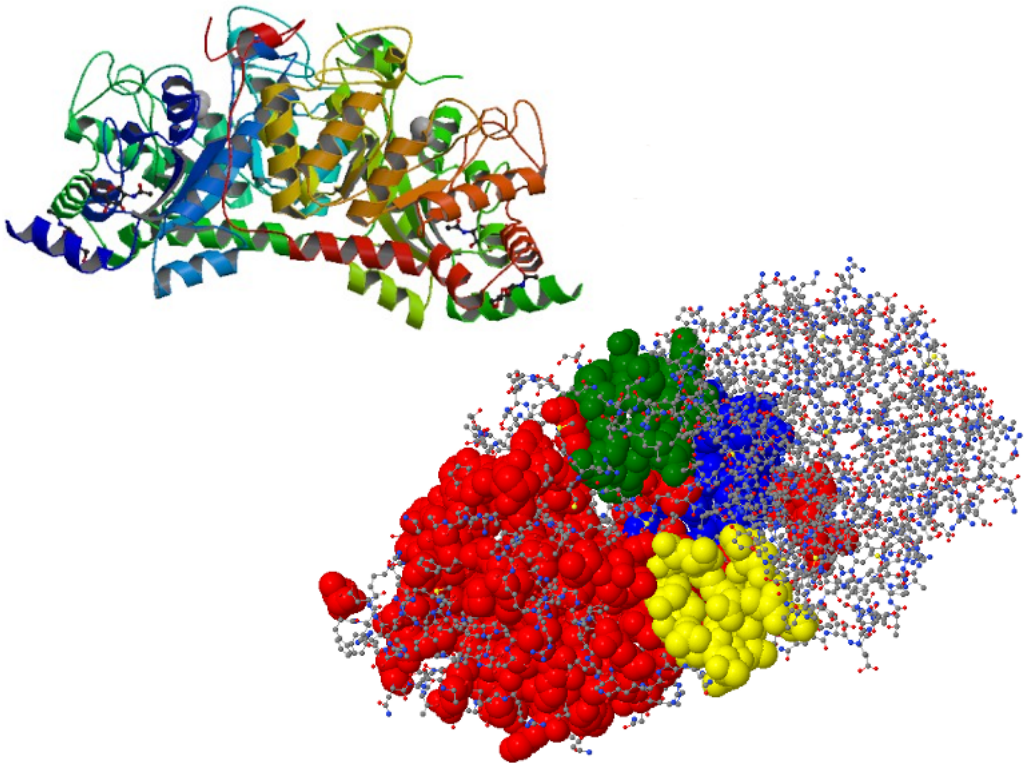


Figure 7.8. Visualization of predicted binding sites for 1ITQ protein. At the left upper corner, an original structure is drawn with ribbons. Predicted binding sites are colored at the bottom right as follows: red - Consurf results, blue - CastP results, yellow - PocketFinder results and green - QSiteFinder results.

have been integrated with usage of general wrapper, which was created for this purpose as described in section 4.3.2. For performing comparison of these services, which was the key reason for integration in the virtual laboratory, a *ResultsConverter* gem, which is responsible for creating a common results' format, has been developed.

The experiment, that operates on pdb files from a single directory, was executed for analyzing *hydrolases* as a large set of proteins. A TaskListCreator object was used for resuming computation if an experiment has failed due to problems with services. Each time a set of files from every service generated for analyzed files and a set of results converted to comparable format was created. Additionally, a script for visualizing the residues that were recognized as binding sites was created. Sample results of visualization of human reneal dipeptidase (1ITQ) protein in Jmol software is presented in Fig. 7.8. Detailed analysis of services comparison is presented in [57]

Chapter 8

Microarray Data Analysis

An experiment created for performing microarray data analysis is described in this Chapter. Firstly, an overview of microarray technology is presented. Afterwards the system for downloading microarray datasets and creating new datasets as well as the clustering library that operates on created data model are introduced. Finally, a structure of the created experiment is shown.

8.1. Problem Description

Microarray technique is an approach that allows to perform a **transcriptome** analysis. Transcriptome is a complete set of cell's mRNA molecules. Microarray is a small chip covered with thousands or tens of thousands of spots. Two microarray technologies are the most popular: oligonucleotides microarrays, made in silica technologies, and spotted microarray, made from glass and covered with a special chemical substance, that allows DNA hybridization process. Spots contain nucleotide probes for hybridize complementary DNA sequences. Analyzed sample has to be marked with special fluorescent dye, green and red dyes are used. Microarray is placed into prepared solution for several hours. DNA from analyzed sample hybridizes to probes on microarray spots. The probes may be built from short oligonucleotide sequences, that consist of 25 nucleotides. In this case gene identification is performed by using many short fragments of every gene that are placed in microarray. Gene expression is marked only if all spots for a particular gene have bound to DNA sequences. The second type of microarray probes are relatively long cDNA sequences of 500 - 2000 basis length. This type is used in spotted microarrays, just one spot is required for marking gene expression. Despite the fact that microarray chip is small, a complete set of genes for the specified organism may be often placed on a single chip, it depends on an organism's genome

length. Thanks to usage of fluorescent dyes, the gene expression in analyzed sample may be recognised. Microarray scanners read amount of dye from the spots. Various techniques are used in order to the make process more reliable. Instead of reading only one type of fluorescent dye, a red to green fluorescent intesity ratio is computed. Additionally, log scale is used in order to normalize induction / repression regions. Finally, using different chemical dyes may influence on gene expression value. Performing microarray analysis on the same sample with two different dyes and computing gene expression on the basis of difference between values for every dye is used.

Microarray experiments are large-scale techniques. Huge amount of data is produced in every experiment. Organisms' genomes consist of thousands of genes, in every microarray experiment many chips are used for marking gene expression in a different time for the same specimen or analysis of a group of species. Various techniques, such as cluster searching and analysis by using clustering algorithms, especially hierarchical clustering, PCA method and Data mining are used during microarray data analysis process.

Gene expression analysis with microarray technique has been applied to a wide variety of important biological problems including mapping metabolic pathways, tissue typing, environmental monitoring and answering a wide range of questions pertaining to medical diagnosis of disease states [41]. Gene expression time series may be helpful in farmacogenomics, that allow medicians to choose treatment with maximized the efficacy with using inormation about individuals' expression patterns change in response to different techniques. Gene expression patterns also are useful in searching for gene function and with comparison to reference values in patient's diagnose process.

Microarray experiments results are available over the Internet. Data are grouped into experiments, with assigned experiment identifier and samples identifiers. Experiments are annotated with organism name. In the virtual laboratory a set of gems and experiments have been developed. Researcher may conduct microarray data analysis with knowledge only on required data identifiers. Analysis may be executed on available datasets or new datasets, created from selected samples for the same organism type and microarray type. Finally, microarray data clustering and visualization may be performed.

8.2. Microarray Analysis in Virtual Laboratory

Gems available in the virtual laboratory are divided on two groups: a set of classes to retrieving data from databases and converting data into object model and a set of implemented clustering algorithms.

8.2.1. Datasets creation

Data analysed inside microarray gems are organised in object model. The data model allows to easily adding new data sources by creating new classes that realize the defined interface. Among data the model concepts we can list:

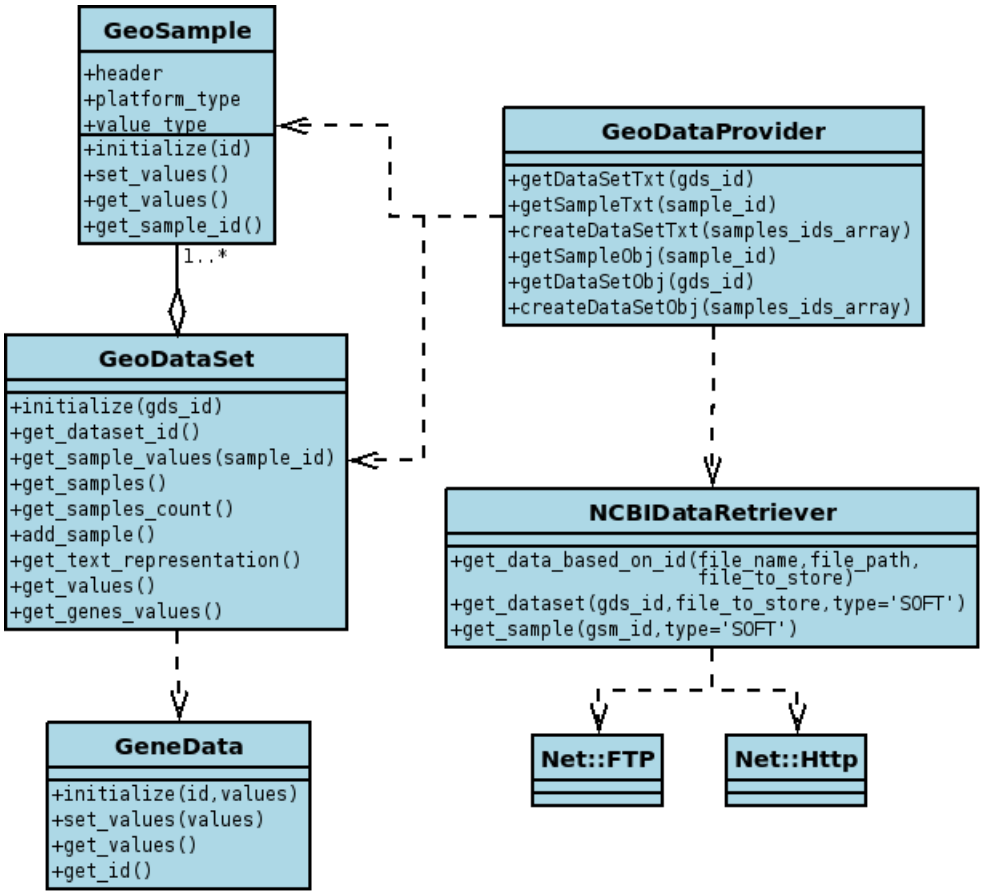


Figure 8.1. Microarray data model class diagram. GeoDataProvider uses data retriever to download dataset from appropriate database or to create a new one from samples identifiers.

Sample is consisted of all gene expressions that can be read from one microarray chip. Data are annotated with sample identifier, platform identifier (microarray type and producer), stored value type and organism name on which analysis was performed.

Dataset is a set of *Samples* created within one microarray experiment. Every sample have to be the same platform type as complete dataset type, also stored value types have to be agreed. New dataset may be created if these limitations are taken into consideration. Datasets are created from samples identifiers.

Gene expression are data transformed from the set of samples into one array that describes one gene expression in every sample. Additionaly it contains information that allows gene identifying.

Data available in databases are often provided in text fromat with well defined structure. Because text format is rather difficult to operate on, data are converted into object data model. The structure of classes belong to this model is presented in Fig. 8.1. Microarray data model consists of following classes:

GeoSample *Sample* equivalent. Identifier is set when object is created with value passed as constructor parameter. Provides methods to access to microarray data and description. *GeoSample* object is created by the data provider. *GeoSample* objects may be analysed separately or as a set of samples inside *GeoDataSet*. For NCBI database *GeoSample* is created from information parsed from HTML page with appropriate *Sample* identifier location.

GeoDataSet Set of samples, microarray experiment. *GeoDataSet* is created by adding *GeoSamples* to it. First added *GeoSample* determines microarray platofrm type and value type of whole dataset. It is required that every next added sample have the same value of both these parameters. *GeoDataSet* provides methods that allow user to get information about dataset's samples: number of samples in dataset; gene expression for complete dataset and for specified sample, which is identified by *sample_ id* identifier; printing dataset in text format, compatibile with a NCBI database dataset formats; getting gene expression information by creating list of *GeneData* objects.

GeneData is a object that contains information about one gene expression profile in a complete dataset. If we can recognize *GeoDataSet* as a matrix, *GeoSample* is one column in matrix values and *GeneData* is one matrix row.

A **GeoDataProvider** class is an interface that allows to access to data. This object provides methods for downloading data from handled databases and returning these data in text fromat. Data from existing datasets are downloaded directly, but for newly created ones and for single sample are created from information available in database. The second group are methods for downloading data and converting their textual form into classes described above. Data provider uses communication object for handling appropriate database. For NCBI database it is *NCBIDataRetriever* class. *DataRetriever* is able to download two types of gene expression values. SOFT data are preprocessed and normalized and can be directly passed to further analysis. RAW data are values directly read from microarray scanner, before next steps of analysis process some normalization and statistical computation must be performed on these data. SOFT data are main type on which computation in the virtual laboratory's experiments will be performed. When *DataRetriever* operates on complete datasets available in database, it uses FTP protocol to download archive with text data files. A path to the dataset archive is created on the basis of dataset identifier. When archive is downloaded, it is unpacked and files with data are parsed. When single Samples are downloaded, *DataRetriever* uses HTTP protocol and the HTTP response with the complete HTML page is parsed. The address of the web page is created on the basis of the sample identifier. When a new dataset is created a single sample approach is used, because new dataset is built by adding samples to it. The sequence diagram for creating new dataset process is presented in Fig. 8.2. Requesting new dataset to be created by *GeoDataProvider* is completed by following steps:

1. Creation of the new *GeoDataSet* object with specified identifier,
2. For every sample identifier that was passed as parameter array do:
   a) Create *GeoSample* object with specified sample identifier,

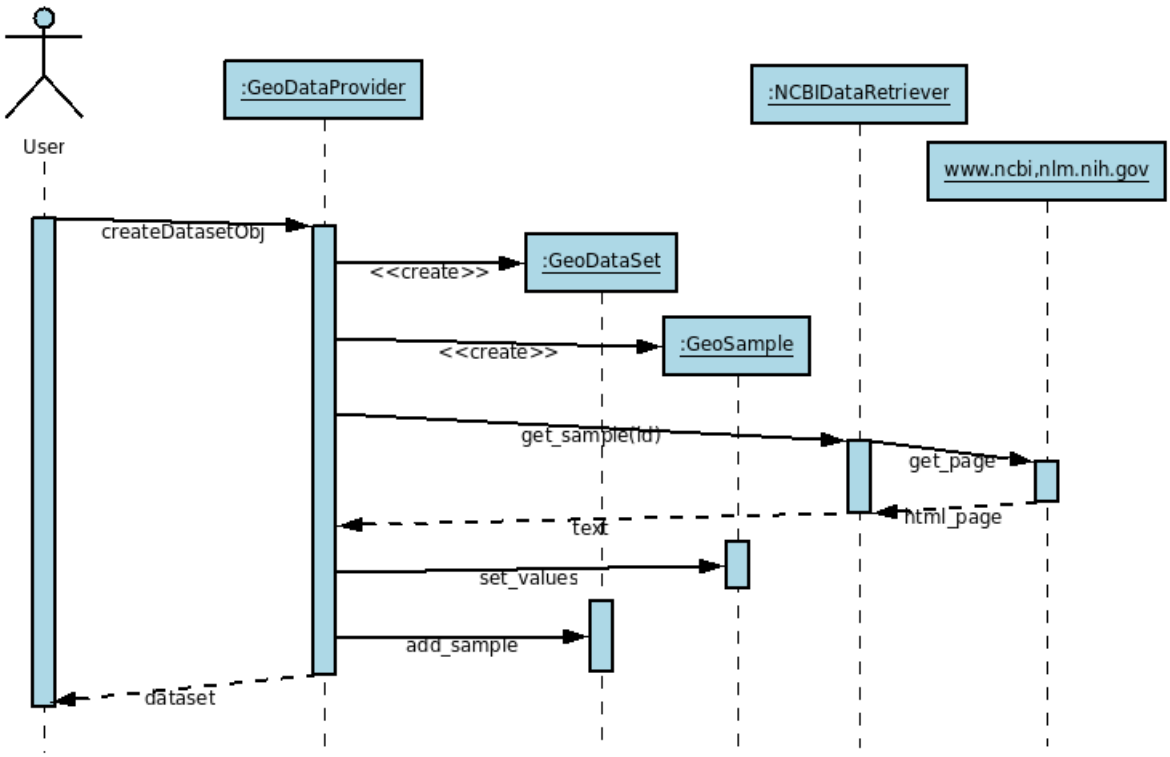


Figure 8.2. Sequence diagram for creating a new dataset from samples identifiers for NCBI database. Samples objects are created on the basis of Web pages and are added to the GeoDataSet object.

b) Call method on *NCBIDataRetriever* for downloading information about sample,
c) Communicate with Web page and download information. The Web page address is generated on the basis of the sample identifier,
d) Return result in a text format,
e) Parse result within *GeoDataProvider* object. Extract information about microarray type, expression values type and gene expressions,
f) Set values to *GeoSample* object,
g) Add a new *GeoSample* to *GeoDataSet* object,

3. As a result new GeoDataSet object is created from separate samples.

Class names prefix "Geo" stands from Gene Expression Omnibus[1].

8.2.2. Data clustering

A library with functions to clustering data, that operate on Geo* objects has been developed and provided in the virtual laboratory. Two types of clustering algorithms are available: agglomerative clustering and Isodata[2] algorithm.

[1] www.ncbi.nlm.nih.gov/geo/
[2] http://fourier.eng.hmc.edu/e161/lectures/classification/node12.html

Hierarchical clustering. An agglomerative version [24] of the hierarachical clustering algorithm relies on a distance matrix between elements. Two nearest elements are joined into one cluster iteratively. A dendrogram that depicts complete clusterisation process (joining all elements into one cluster) is created. A threshold value for a distance between clusters that cannot be joined together provides to creating more than one cluster as a result. Clustering process is dependent on elements' distance metrics, linkage criteria (counting distance between clusters) and a way in how cluster is represented.

Isodata algorithm. Isodata stands for Iterative Self-Organizing Data Analysis Technique. It is more sophisticated version of K-means clustering algorithm. This algorithm requires on start a K value, that means expected cluster number to create, the same as in K-means algorithm, but resulted cluster numbers may be different from the expected value. Isodata algorithm is able to split and merge the created clusters, these operations depend on the number of elements that are assigned to each cluster and cluster structure. Centers of clusters are choosen from the data set at the beginning of algorithm execution. After that, each sample is assigned to the closest cluster center, on the basis of a distance counted with the usage of selected samples metrics. Clusters with too few members are then discarded. If currently too few clusters are created (threshold value is *K/2*), clusters are split. This step is done on basis of a standard deviation of the distance from every sample to the cluster center, and a number of elements assigned to each cluster. In the case of too many clusters created, they are merged on the basis of a distance matrix between cluster centers. Isodata is an iterative algorithm. For clusters created in every loop, the samples are asigned at the loop begining and the full process is iterated. The biggest problem in using Isodata algorithm is defining set of parameters required to performing clusterization. Required parameters are:

- number of clusters desired,
- maximum number of iterations allowed,
- maximum number of pairs of clusters which can be merged in a single iteration,
- a threshold value for minimum number of samples in each cluster can have (used for discarding clusters),
- a threshold value for standard deviation (used for split operation),
- a threshold value for pairwise distances (used for merge operation).

This number of parameters to set requires having large knowledge about the data. If we are clustering microarray data, setting these parameters properly is not always possible.

Library implementation. A class diagram for the created clustering library is presented in Fig. 8.3. Four modules are included in this library:

Data model. Classes that have information about cluster, samples assigned to cluster and cluster representation.

- **Cluster** - Contains all samples asigned to cluster, allows adding and removing samples. Provides information about cluster score. Important cluster element

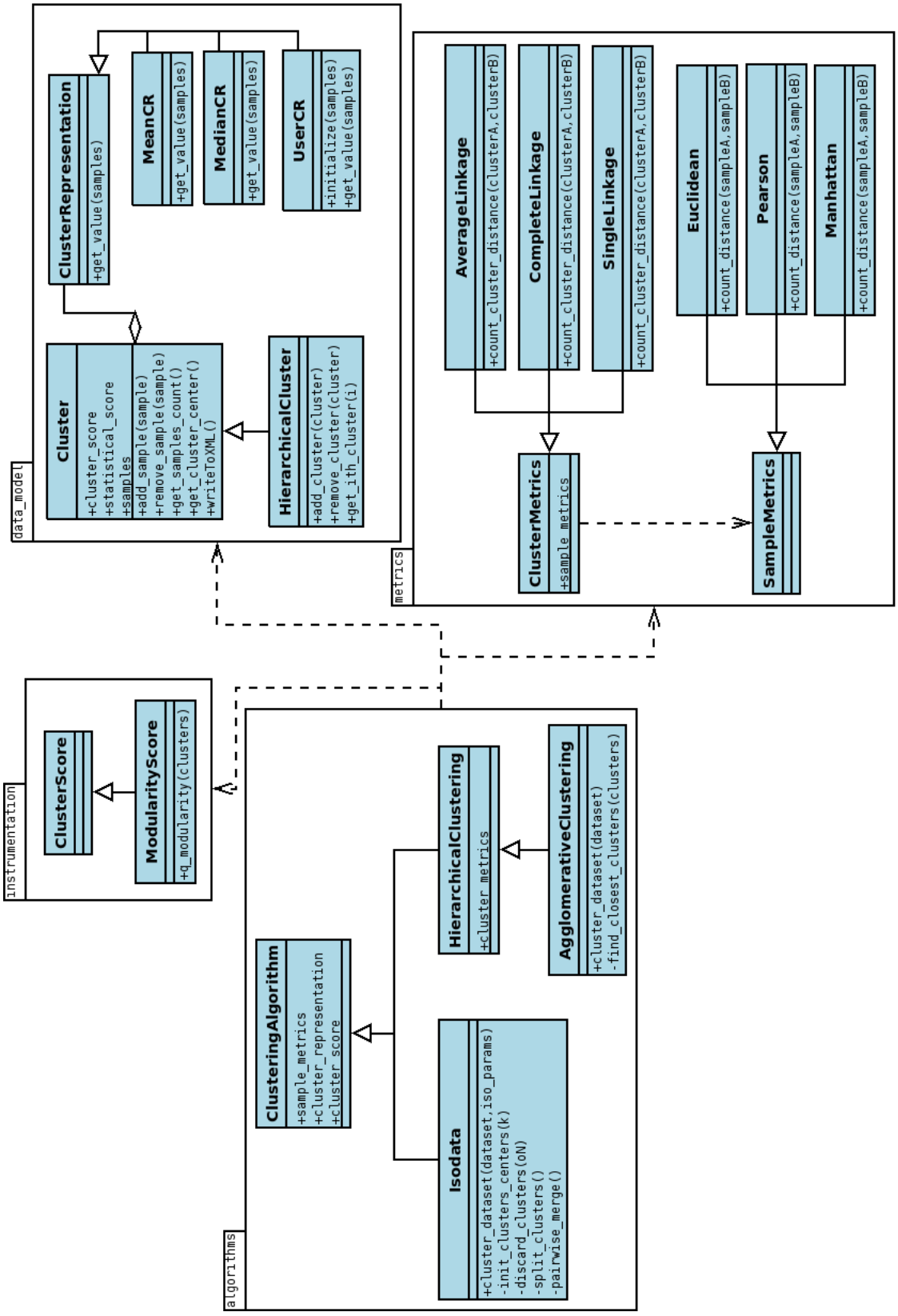


Figure 8.3. Class diagram for microarray data clustering library.

is its representation, which is a cluster value available to external usage. Additionaly, a method for writting cluster state, and its all samples to XML format is created.

- **HierarchicalCluster** - It is an extension of *Cluster* class, that allows joining other clusters into this cluster, in order to create clusters hierarchy. This class provides also methods for managing included clusters and redefines methods for access to samples. How this method is recursively called on clusters included in the main cluster, and then generated results are gathered.
- **ClusterRepresentation** - Cluster representation is based on selecting one element from samples assigned to cluster, on which further computation will be performed. Three types of cluster representation sub class are available: median (middle element), mean (mean from all elements, computed in every dimension) and user (used in Isodata algorithm, representation is set externally and it is independet of samples asigned to cluster).

Metrics. A way how data are clustered is dependent on selected metrics. Two type of metrics are used in library:

- **Linkage criteria** - it is a metric between clusters. It determines the distance between a set of samples as a function of a two samples distance. Available criteria are:
  — Complete Linkage ($\max\{d(a,b) \ : \ a \in A, \ b \in B\}$),
  — Single Linkage ($\min\{d(a,b) \ : \ a \in A, \ b \in B\}$) and
  — Average Linkage (UPGMA, $\frac{1}{|A||B|}\sum_{a\in A}\sum_{b\in B} d(a,b)$).
  A selected sample metric is used for counting linkage value. Sample metric is set when Linkage criteria object is created.
- **Distance Measurements Between Data Points** - available metrics are: Euclidean, Manhattan and Pearson Correlation.

Algorithm instrumentation. Clustering algorithms may perform computation of clustering score on the basis of information provided by classes belonging to *instrumentation* module. Currently available measure, modularity score, provide information about reasonability of further joining examples into clusters.

Algorithms. Two different algorithms are developed in clustering library:

- **Agglomerative clustering** - This algorithm creates *HierarchicalCluster* objects. At the begining every example creates single cluster. In each subsequent iteration the two closest clusters, in the sense of selected metrics, are joined into one. Both *Linkage criteria* and *samples distance metrics* have an influence on clusters similarity. Clusters grouping is broken when the distance between clusters exceed a threshold value. An array of *HierarchicalCluster* objects is returned as a clusterization result. *HierarchicalCluster* objects may be reviewed recursively, every cluster contains its sub-clusters.
- **Isodata** - An algorithm that operates on *Cluster* objects. Any of available Samples metrics may be used. An array of *Cluster* objects, with assigned samples, is returned as a result.

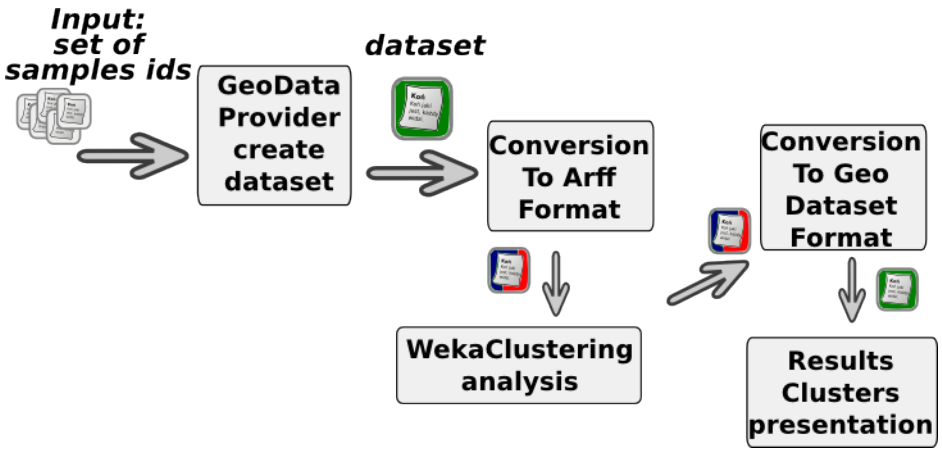


Figure 8.4. The structure of microarray data analysis experiment. Dataset is created from sample identifiers. Conversion to ARFF format is required for analysis in Weka library. Results are converted back to the GeoDataset.

A way how data model used in library and clustering algorithms are designed allows user to cluster *GeoSample* objects as well as *GeneData* objects. Clustering samples may be useful in assigning a new patient to one of reference gene expression values, while clustering genes may be useful in analysing gene function and drugabillity process.

8.3. Sample Microarray Experiment in Virtual Laboratory

The created experiment presents how to use data provider, format converters and perform data clusterization. The model of the experiment is depicted in Fig. 8.4. A set of sample identifiers from NCBI Gene Expression Omnibus database is an input dataset. These identifiers are used to create *GeoDataSet* object within *GeoDataProvider* gem. Because Weka library is used to perform clusterization process (gems that provide Weka functionality), a *GeoDataSet* object has to be converted to ARFF fromat, in order to use these data within *WekaClusterer* gem. A Cobweb algorithm is used in clusterization process. As a result of clusterization, ARFF file is returned. It is necessary to convert these result back into GeoDataSet object. The last step of this experiment is results visualization.

8.4. Summary

The created solution provides a possibility of analysis microarray data using one of the available clustering technologies. Three clustering libraries (Cluto, Cluster and Weka) have been integrated and may be used with data format converters. Additionaly a clustering library that operates directly on microarray data model has been developed. The developed data model and data providers allow researcher to analyse currently available microarray datasets as well as to create new datasets by joining samples from different experiments. The data may be obtained from NCBI

GEO or EBI ArrayExpress databases, but new databases may be easily added by implementing defined data retrieval interface.

Chapter 9

Results Presentation Layer

This chapter presents the applications that are often used in the last stage of an experiment. The visualizers are used by researchers in the results analysis. In the virtual laboratory numerical data may be plotted with the usage of Gnuplot program, which is presented first. Then, applications integrated with Java Web Start technology for protein structure visualization, like Jmol and ProteinWorkshop, and Jalview for visualizing sequence alignments are introduced. Finally, the integration of a microarray clustered data visualization with JavaTreeView is presented.

9.1. Plotting Numerical Data With Gnuplot

Gnuplot is a command-driven interactive function and data plotting program. Gnuplot can produce output directly on screen, or in many formats of graphics files, including PNG, EPS, SVG, JPEG and many others. The program can be used both interactively and in batch mode using scripts. The second approach is used to run it in the virtual laboratory. Plotting two-dimensional functions and data points in many different styles (points, lines, error bars) and plotting three-dimensional data points and surfaces (also in many different styles, like contour plot, mesh) as well as algebraic computation in integer, float and complex arithmetic are possible in gnuplot. This program is used as the plotting engine of *GNU Octave*, *Maxima* and *gretl*, and it can be used from various scripting languages, including Ruby, via *Ruby Gnuplot*.

The integration of the gnuplot was performed by wrapping its binaries and publishing as a Web service. Two methods are accessible: `plot`, for drawing a single serie plots, and `multiplot`, where data from many series may be included. Both

methods take two arguments: a gnuplot script and input data. In the first case, an input is a single file, while in `multiplot` method it is an array of data files. Both methods are organized as follows:

1. Create temporary directory for handling plotting request,
2. Store input data file (many files in `multiplot`) in this directory,
3. Analyse script body, replace the *'input'* string in the script with correct name of created file (in `multiplot` method replaced string patterns are *input{NR}*, where *{NR}* is the next number that comes out of the length of input data array),
4. Set the name of results plot in the script,
5. Store corrected script in the temporary directory,
6. Execute *gnuplot*,
7. Remove temporary directory,
8. Read result file, if the file was correctly created its URL path is returned, in the other case an information about error is the result.

The usage of *gnuplot* scripts directly, instead of *gnuplot* wrappers, like Ruby Gnuplot, enables researchers to use the complete gnuplot capabilities. An example script and the results obtained for this script execution are presented in Fig. 9.1. This is the result of execution an Protein Sequence and Structure comparison experiment (see Chapter 6) on particular data.

9.2. Protein Structure Visualization with Jmol and ProteinWorkshop

Protein structure visualization is a quite complex task, but the programs that realize it aid researchers working on different aspects of protein structure. The visualization is often based on the information from pdb file. The variety of available solutions give a plenty of functions for showing structures, like different representation of structure (e.g. spacefill, balls and sticks, cartoons, ribons, surfaces), different coloring schemes (e.g. by atom type, basing on secondary structure, by group or other values from pdb file), selecting atoms or measurements. From the most popular structure visualizers two Java-based applications, Jmol and ProteinWorkshop, were selected and integrated into the virtual laboratory. The integration was performed with Java Web Start Technology.

Java Web Start is a technology that enables deploying standalone Java software applications over the network. It uses The Java Network Launching Protocol & API (JNLP), which provides a browser-independent architecture for deploying Java 2 technology-based applications to the client desktop. Application must be packaged in JAR files to work with Java Web Start and correct JNLP file must be provided to make an application accessible. The JNLP file is an XML document which is outlined with *'jnlp'* as the root element and *'information'*, *'security'*, *'resources'*, and *'application-desc'* as its subelements. The base path from which application should be downloaded is defined as *'codebase'* in *'jnlp'* element. The *'information'* element defines application title, vendor and description. Each application is, by

```
set pm3d at b
set palette defined ( 0 1.0 0.0 0.0, 0.5 1 1 1, 1 0 0 1)

set multiplot layout 4,1 title "W Score profiles for PROTEIN_NAME and chain PROTEIN_CHAIN"
unset title
#Plot 1 - sequence alignment
set ylabel "AA Sequence"
splot "input0" t ""

#Plot 2 - structural code alignment
set xrange [ 1.0000 : (SAMPLES_COUNT - 2) ] noreverse nowriteback
set ylabel "Structural codes"
splot "input1" t ""

#Plot 3 - 3D structures alignment
set xtics border
set xlabel "residue"
set xrange [ 1.0000 : SAMPLES_COUNT ] noreverse nowriteback
set ylabel "3D Structure"
splot "input2" t ""
```

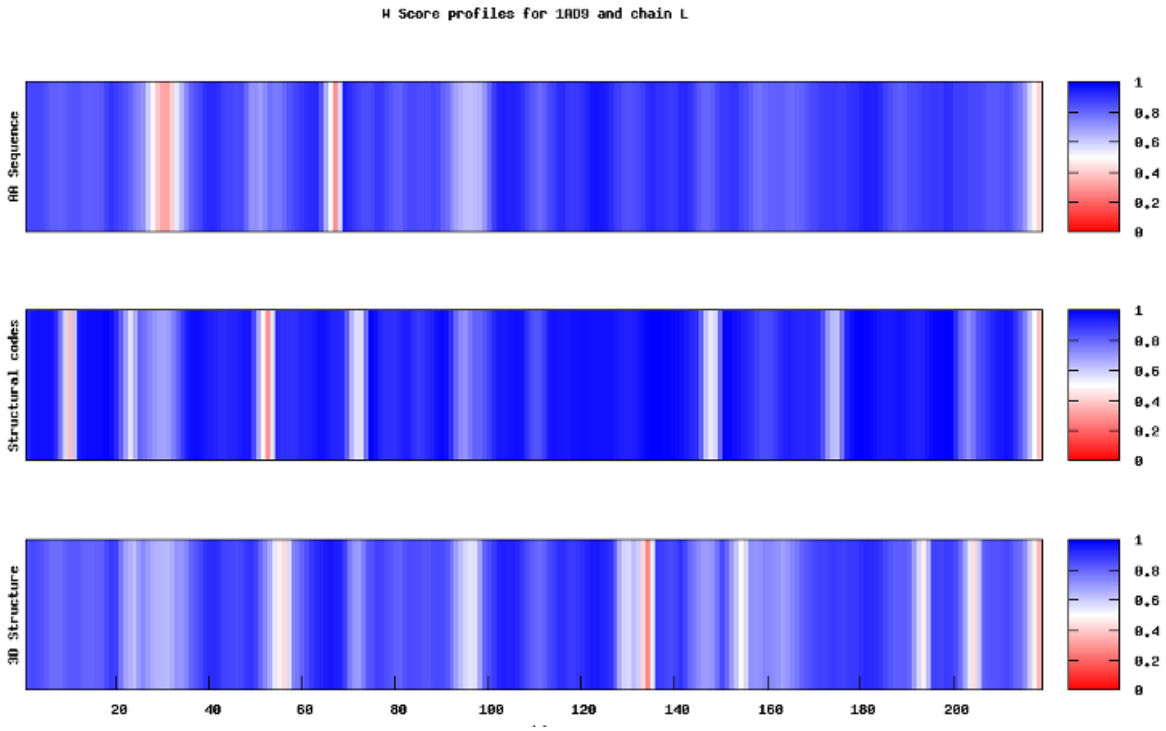


Figure 9.1. An example of the gnuplot script and the plot. This is a part of the script used to generate multi series plot in protein sequences and structures comparison experiment. The results presents W profiles for L chain of Igg-Fab fragment of engineered human monoclonal antibody CTM01protein (structure ID: 1AD9), obtained for ALL families comparison. See details in Chapter 6.

default, run in a restricted execution environment, similar to the Applet sandbox. The *'security'* element can be used to request unrestricted access. The '*resources*' is used to specify all the resources, such as Java class files, native libraries, and system properties, that are part of the application. All required jar files to run application, listed in this section, must be accessible from the same server as it was stated in *'codebase'* value. The '*application*' indicates that the JNLP file is launching an application (as opposed to an applet). This element has an optional attribute, *'main-class*', which can be used to specify the name of the application's main class. Arguments can be specified to the application by including one or more nested *'argument'* elements.

9.2.1. Protein Workshop

The Protein Workshop [52] is a tool for quickly generating high quality images of large macromolecular structures. The workshop provides easy-to-use controls for manipulating structural presentation based on features such as hydrophobicity, residue and conformation. This software is based on the Java2 platform and the graphics library uses JOGL; to run this software the platform must support OpenGL. Protein Workshop is deployed in Protein Data Bank using Java Webstart. Its integration with the virtual laboratory is based on modified JNLP file. When the user runs Protein Workshop application, it is downloaded from its provider servers (www.pdb.org), but an URL to analyzed file is modified. The appropriate URL from the virtual laboratory or location on the user's disk is inserted as an '*structure_url*' argument. Additionally, the options that Protein Workshop handles may be set. Multiple structures (defined in separated pdb files) also may be loaded at once. An example of the structure visualization rendered in the Protein Workshop software is shown in Fig. 9.2.

9.2.2. Jmol

Jmol [5] is an open-source Java viewer for three-dimensional chemical structures, with features for chemicals, crystals, materials and biomolecules. Features include reading a variety of file types and output from quantum chemistry programs, and animation of multi-frame files. It consists of three main components: a web browser applet, a standalone Java application and an integratable Java component. Rendering in Jmol may be driven by scripts, which is one of the its advantages. In the virtual laboratory Jmol is accessible in two ways, presented below.

Simple Jmol application, available to download from the virtual laboratory Http server with using Java Web Start. It may be used to execute one Jmol script. An example of its usage is an experiment for comparing binding site prediction services, where the researcher wants to show only one of obtained result (and generated Jmol scripts for their visualization), or every of protein should be visualize in a separated Jmol application. As a main class the `org.openscience.jmol.app.Jmol` is used. A path to the visualization script is inserted as an argument in JNLP file.

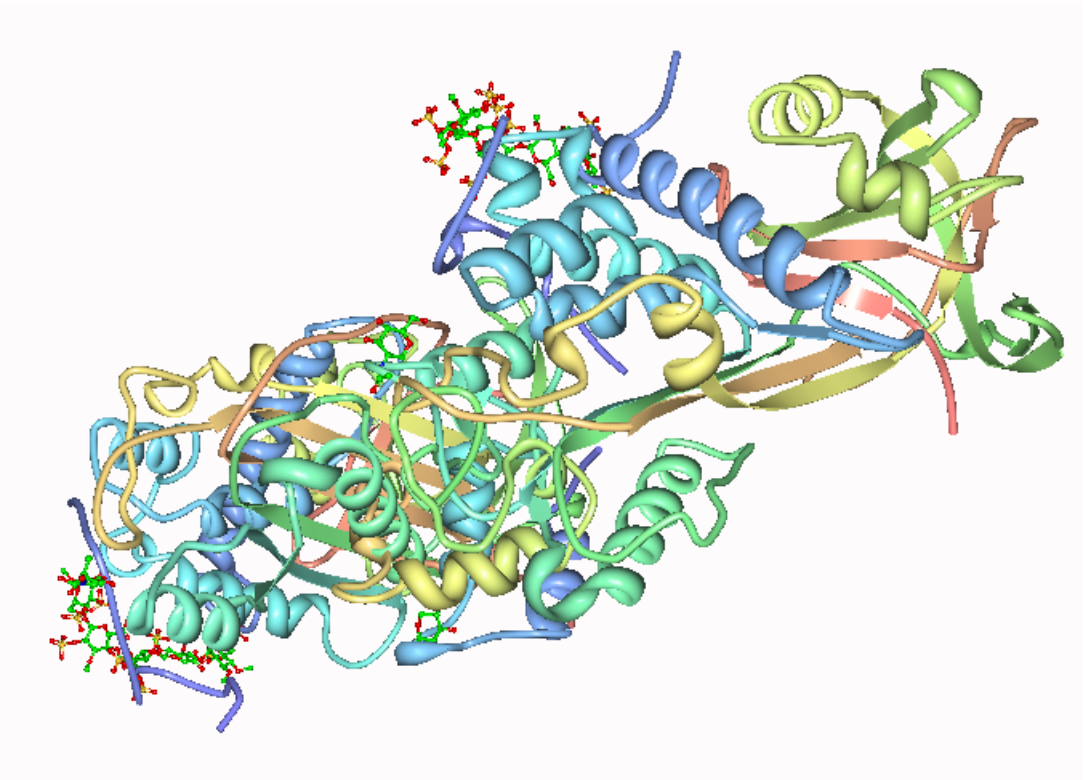


Figure 9.2. The Protein Workshop visualization example.

An example of protein structure visualization performed in Jmol is shown in Fig. 7.8.

Enhanced Jmol visualizer, has been developed, which embeeds Jmol viewer and extends it capabilities with possibility of changing visualized structures and dynamically selecting residues that were predicted as binding sites in the different services without necessity of using Jmol scripts. This program requires a path to XML file with result definitions. In this file all information about analyzed proteins have to be included. The example of input file is presented on Listing. A *'structure'* element defines its name, which is displayed in a listbox, a path to pdb file which has to be opened. Every result is defined in *'service'* element, which also has *'name'* attribute that is displayable. Residues numbers are listed in *'chain'* elements, they are used to perform selection in Jmol viewer.

```
<inputs>
  <structure>
    <name>PDB_ID_1</name>
    <path>path/to/pdb/file</path>
    <service name="SERVICE1">
      <chain id="A">
        1 2 3 4 5 6
      </chain>
      <chain id="B">
        37 39
```

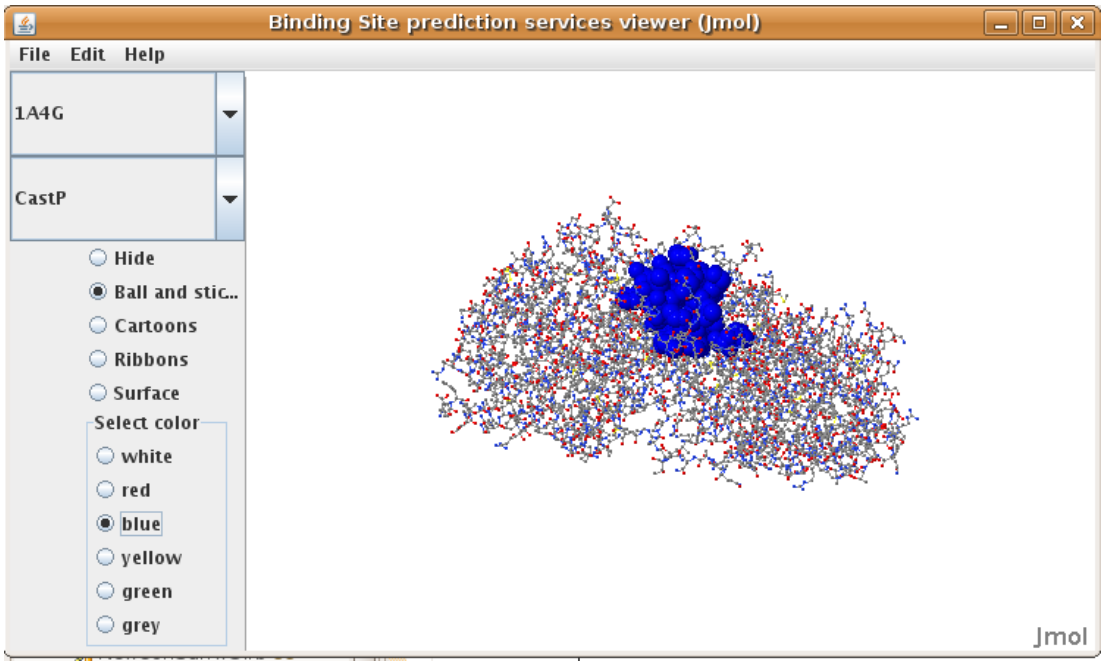


Figure 9.3. Enhanced Jmol viewer main window.

```
      </chain>
      ...
    </service>
    <service name="SERVICE2">
      ...
    </service>
    ...
  </structure>
  <structure>
    ...
  </structure>
  ...
</inputs>
```

The application window is presented in Fig. 9.3. On the left panel there are available combo boxes for selecting structure to display and coloring results. The complete protein may be displayed in one of available representations: balls and sticks, cartons, ribbons or surface. The results for selected service are always displayed as coloured spacefills (available colors are white, red, blue, yellow, green and grey) or may be hidden. Results from many services may be displayed at the same time, but only one protein may be examined at once. If selected protein is changed, the visualization settings from last available protein are lost.

9.3. Sequences Alignment in Jalview

Jalview [67] is a Java multiple alignment editor and analysis tool. Its primary function is the editing and visualization of sequence alignments, and their interactive analysis. Tree building, principal components analysis, physico-chemical property

conservation and sequence consensus analyses are built in to the program. Web services enable Jalview to access remote alignment and secondary structure prediction programs, as well as to retrieve protein and nucleic acid sequences, alignments, protein structures and sequence annotation. Sequences, alignments, trees, structures, features and alignment annotation may also be exchanged with the local filesystem. Multiple visualizations of an alignment may be worked on simultaneously, and the user interface provides a comprehensive set of controls for colouring and layout. Alignment views are dynamically linked with Jmol structure displays, a tree viewer and spatial cluster display, facilitating interactive exploration of the alignment's structure. The application provides its own Jalview project file format in order to store the current state of an alignment and analysis windows[1].

Jalview may be obtained with Java Web Start. It is integrated in the virtual laboratory with the same technology. Defined JNLP file downloads Jalview from its provider website ('http://www.jalview.org/webstart' as *'codebase'*). A path to the file with generated alignments is set as *open* argument's value. Additional arguments, like coloring scheme, defaultly defined as Clustal coloring in JNLP file, may be set. As a main class, `jalview.bin.Jalview` is used. An example of Jalview's alignment visualization is presented in Fig. 6.4.

9.4. Microarray Clustering Results in JTreeView

Java TreeView is an open source, cross-platform gene expression visualization tool. It may be used for interactively displaying clustered gene expression data. Tree view uses three files for displaying clusterization results: a .cdt (clustered data table) file, which contains the original gene data (gene and samples names and expressions), but reordered, to reflect the clustering, a .gtr file (gene tree), and a .atr file (array tree). These tree file reflect the history of how the cluster was built, and can be used to contruct how the tree(s) should look.

JTreeView is integrated in the virtual laboratory with Java Web Start technology. The paths to *cdt*, *gtr* and *atr* files are execution attributes. The results have to be converted from used data model (described in section 8.2.2) and their XML represenations to the files used by JTreeView (cdt, gtr and atr files).

9.5. Summary

In the scope of this thesis the wide range of applications that are able to visualize data have been integrated in the virtual laboratory. Most of them are available using Java Web Start technology. There are applications for visualizing specialized data, such as protein structure (ProteinWorkshop, Jmol), sequence alignments (Jalview) or gene expression (JTreeView) as well as simple numerical data (Gnuplot). Numerical data may be also displayed with using R scripts and the R gem capabilities. The visualizers are often complex applications and may be used not only for presenting data obtained by experiment executions, but also for analyzing these data.

[1] This description comes from JalView documentation

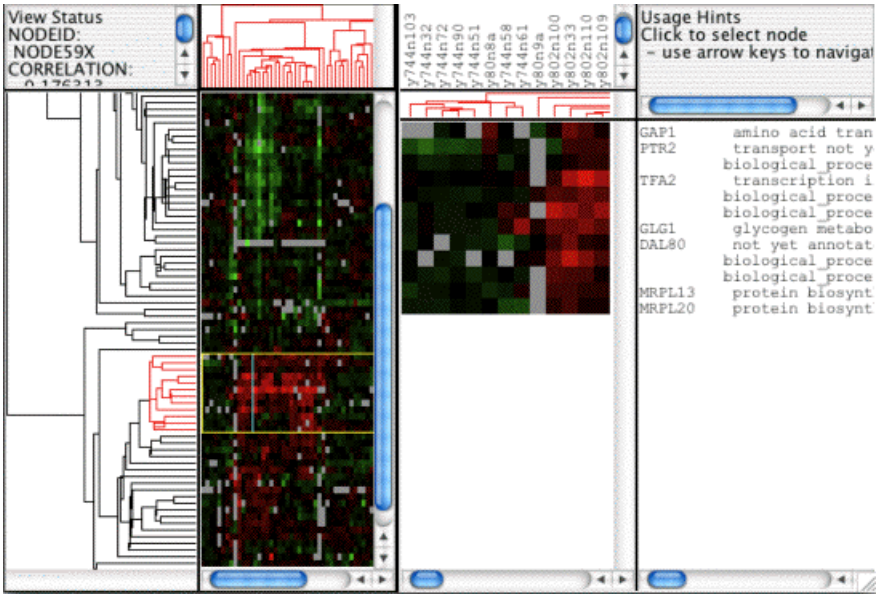


Figure 9.4. Gene expression visualization in JTreeView software.

Chapter 10

Conclusions and Future Work

This Chapter summarizes the thesis goals that were achieved during the work on it and presents tasks recognized as future work to enrich the virtual laboratory as a bioinformatics framework.

10.1. Summary

The main goal of this thesis was to prepare a set of applications suited to solve common problems in the field of bioinformatics, as well as to integrate them into the virtual laboratory. This goal has been successfully achieved. The applications were integrated and registered in the virtual laboratory. The experiments that use prepared applications were developed.

All the achieved sub-goals, that were identified in Chapter 1 are listed below:

- **Analysis of the available bioinformatics applications** - as a result of this analysis the main bioinformatics research areas to be supported were selected. Additionally, bioinformatic databases that are required to obtain needed data were identified.
- **Classification of the bioinformatics applications and their division into a number of categories**. Two classifications of applications have been developed: by scope of usage and by technology. On the basis of the first one, the complete set of applications that were integrated into the virtual laboratory has been chosen.
- **The design of integration of the selected applications.** On the foundation of the available grid object implementation technologies and selected applications, two integration mechanisms were developed, as well as to every application an appropriate integration technology were assigned.

Technology	Gems
Local gem	4
Implementation	6
Libraries integration	4
Binary program wrappers	8
HTTP service wrappers	8
Java Web Start applications	4
External Web service	8

Table 10.1. Created gems statistics, classification by technology

- **Creating a set of ViroLab gems and preparing experiments.** The presented sets of gems and experiments were created. The most advanced bioinformatic experiments are protein sequence and structure comparison as well as a comparison of ligand binding site prediction services. In every experiment gems responsible for performing main computation parts may be substituted by another application (algorithm) that solves the same problem.
- **Preparing general methods and tools to make using the bioinformatics applications easier in the virtual laboratory experiments.** The created integration mechanisms facilitate adding new applications to the virtual laboratory. In the other hand, some additional gems, like data format converters, were created for ensuring correct cooperation between gems in the same experiment.

During the work on this thesis the following artifacts were created:

- Two new mechanisms: task queuing system and binary program wrapper, that are extensions used for integration new applications,
- 42 gems suited to solve bioinformatics problems,
- two main experiments: protein sequence and structure comparison and a comparison of ligand binding site prediction services,
- some additional sample experiments, where gems usage were explained.

The created gems may be divided into categories, according to classifications presented in section 4.2: by gem technology (see Figure 4.2), listed in Table 10.1 and by their scope of usage (see Figure 4.3), listed in Table 10.2.

The ViroLab virtual laboratory is a complete environment for running scientific applications. The enrichment of the virtual laboratory capabilities by new gems was performed with usage of existing gem technologies, as well as using two mechanisms that were added during this work. The advantages of using the virtual laboratory as a framework for bioinformatics applications are as follows:

- adding new applications and integrating databases is relatively easy and may be performed for most of bioinformatics applications, irrespectively of technology the application is made,
- thanks to Grid Object model application technology is transparent to the user, so researcher can use any of existing gems in the bioinformatic experiments, especially if there are more applications suited to solve the same type of problem,

<table>
<tr><th>Layer</th><th colspan="4">Subcategories</th></tr>
<tr><td>Results presentation 5</td><td>numerical data 1</td><td>protein structure 2</td><td>sequences alignment 1</td><td>microarray data 1</td></tr>
<tr><td>Specialized analysis 21</td><td colspan="2"><table><tr><td>Protein Comparison</td><td>9</td></tr><tr><td>Sequences align.</td><td>4</td></tr><tr><td>Structures align.</td><td>4</td></tr><tr><td>Additional</td><td>1</td></tr></table></td><td><table><tr><td>Binding Site pred.</td><td>10</td></tr><tr><td>Services</td><td>9</td></tr><tr><td>Additional</td><td>1</td></tr></table></td><td><table><tr><td>Microarray</td><td>2</td></tr><tr><td>Data preparing</td><td>1</td></tr><tr><td>Data clustering</td><td>1</td></tr></table></td></tr>
<tr><td>Basic analysis 11</td><td>Statistics 1</td><td>Clustering 3</td><td>Data mining 5</td><td>Dimensions 2</td></tr>
<tr><td>Database access 5</td><td colspan="2">Database access 4</td><td colspan="2">Conversion 1</td></tr>
</table>

Table 10.2. Created gems statistics, classification by scope of usage

- managing results of the experiment execution is organized, and allows user to access and browse these data easily.

10.2. Future work

The prepared large set of gems and created experiments do not exhausted a possibilities of enhancing the virtual laboratory capabilities. In the future, the following work could be done to improve the virtual laboratory as a bioinformatic framework.

- **New applications integration**. There are many research areas in the bioinformatic field, that are not covered by this thesis. Adding new software that may be applicable on the field of phylogenetic analysis, protein interaction modelling, drug researching, pattern recognition or decission support may enlarge the number of possible users.
- **Integration of applications with Graphical User Interface**. Currently this type of application is integrated with using the Java WebStart technology. The better way to integrate them might be using plugin architecture inside ViroLab user interfaces. The other type of solution may be applets, AJAX, JavaFX that could be applicable in EMI interface.
- **Graphical workbench for using bioinformatics services.** The structure of the general bioinformatic problem, presented in this thesis, shows bioinformatic experiment as a multiple application pipeline. Creating experiments may be easier with using graphical environment to connect gems in appropriate order. This step however requires some additional work, like defining accepted and produced data formats for every gem and creating converters.
- **Integration of a mechanism to run own local command-line applications.** Many researchers use their own programs to create or analyze experiment results. The enhanced binary program wrapper may be applicable to use these

programs during experiment execution in the same way as “standard” gems are used.

These proposed enhancements will enable the further development of virtual laboratory as a convenient and flexible environment for applications of bioinformatics and other research domains.

Bibliography

[1] Bioconductor project homepage, http://www.bioconductor.org/.

[2] Cluto homepage, http://glaros.dtc.umn.edu/gkhome/views/cluto.

[3] Department of Bioinformatics and Telemedicine, Jagiellonian University - Medical College, http://www.bioinformatics.cm-uj.krakow.pl.

[4] H2O homepage, http://dcl.mathcs.emory.edu/h2o/.

[5] Jmol: an open-source Java viewer for chemical structures in 3D, http://www.jmol.org/.

[6] JRuby homepage, http://www.jruby.org/.

[7] Official ViroLab webpage, http://virolab.org/.

[8] R Project homepage, http://www.r-project.org/.

[9] RCSB PDB homepage, http://www.pdb.org.

[10] Ruby language homepage, http://www.ruby-lang.org/.

[11] Virolab EPE homepage, http://virolab.cyfronet.pl/trac/epe.

[12] ViroLab virtual laboratory, http://virolab.cyfronet.pl/.

[13] PDB File Format - Contents Guide Version 3.20, September 2008.

[14] T. Arodz. Bioinformatics course, AGH, 2008/2009.

[15] A. D. Baxevanis and F. B. F. Ouellette. *Bioinformatics: A Practical Guide to the Analysis of Genes and Proteins*. Wiley-Interscience, April 2001.

[16] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne. The Protein Data Bank. *Nucl. Acids Res.*, 28(1):235–242, January 2000.

[17] A. T. Binkowski, S. Naghibzadeh, and J. Liang. CASTp: Computed Atlas of Surface Topography of proteins. *Nucl. Acids Res.*, 31(13):3352–3355, July 2003.

[18] G. P. Brady and P. F. Stouten. Fast prediction and visualization of protein binding pockets with PASS. *J Comput Aided Mol Des*, 14(4):383–401, May 2000.

[19] M. Brylinski, L. Konieczny, P. Czerwonko, W. Jurkowski, and I. Roterman. Early-stage folding in proteins (In Silico) sequence-to-structure relation. *Journal of Biomedicine and Biotechnology*, 2 (2):65–79, 2005.

[20] M. Brylinski, L. Konieczny, A. Kononowicz, and I. Roterman. Conservative secondary structure motifs already present in early-stage folding (in silico) as found in serpines family. *Journal of Theoretical Biology*, 251:275–285, 2008.

[21] M. Brylinski, L. Konieczny, and I. Roterman. SPI–structure predictability index for protein sequences. *In Silico Biology*, 5:0022, 2004.

[22] M. Brylinski, K. Prymula, W. Jurkowski, M. Kochanczyk, E. Stawowczyk, L. Konieczny, and I. Roterman. Prediction of Functional Sites Based on the Fuzzy Oil Drop Model. *PLoS Computational Biology*, 3(5):e94+, May 2007.

[23] M. Bubak, T. Gubala, M. Malawski, B. Balis, W. Funika, T. Bartynski, E. Ciepiela, D. Harezlak, M. Kasztelnik, J. Kocot, D. Krol, P. Nowakowski, M. Pelczar, J. Wach, M. Assel, and A. Tirado-Ramos. Virtual Laboratory for Development and Execution of Biomedical Collaborative Applications. In *Computer-Based Medical Systems, 2008. CBMS '08. 21st IEEE International Symposium on*, pages 373–378, 2008.

[24] P. Cichosz. *Systemy uczace sie*. Wydawnictwa Naukowo-Techniczne, 2 edition, 2007.

[25] E. Ciepiela, J. Kocot, T. Gubala, M. Malawski, M. Kasztelnik, and M. Bubak. Virtual Laboratory Engine GridSpace Engine. In M. Bubak, M. Turala, and K. Wiatr, editors, *Proceedings of Cracow Grid Workshop - CGW'07*, pages 53–58, Krakow, Poland, October 2007. ACC CYFRONET AGH.

[26] S. J. Cox, editor. *Soaplab - a unified Sesame door to analysis tools*, number ISBN - 1-904425-11-9. All Hands Meeting, Martin Senger, Peter Rice, Tom Oinn, September 2003.

[27] J. Dundas, Z. Ouyang, J. Tseng, A. Binkowski, Y. Turpaz, and J. Liang. CASTp: computed atlas of surface topography of proteins with structural and topographical mapping of functionally annotated residues. *Nucl. Acids Res.*, 34:W116–118, July 2006.

[28] R. C. Edgar. Muscle: a multiple sequence alignment method with reduced time and space complexity. *BMC bioinformatics*, 5(1), August 2004.

[29] R. C. Edgar. Muscle: multiple sequence alignment with high accuracy and high throughput. *Nucl. Acids Res.*, 32(5):1792–1797, March 2004.

[30] W. Funika, D. Harezlak, D. Krol, P. Pegiel, and M. Bubak. Developer and User Interfaces to the Virolab Virtual Laboratory. In M. Bubak, M. Turala, and K. Wiatr, editors, *Proceedings of Cracow Grid Workshop - CGW'07*, pages 47–52, Krakow, Poland, October 2007. ACC CYFRONET AGH.

[31] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[32] R. Gentleman, V. Carey, W. Huber, R. Irizarry, and S. Dudoit, editors. *Bioinformatics and Computational Biology Solutions Using R and Bioconductor (Statistics for Biology and Health)*. Springer, 1 edition, August 2005.

[33] T. Gubala, M. Kasztelnik, M. Malawski, and M. Bubak. Development and Execution of Collaborative Application on the ViroLab Virtual Laboratory. In M. Bubak, M. Turala, and K. Wiatr, editors, *Proceedings of Cracow Grid Workshop - CGW'07, October 2007*, Krakow, Poland, 2007. ACC-Cyfronet AGH. to appear.

[34] M. Hendlich, F. Rippmann, and G. Barnickel. LIGSITE: automatic and efficient detection of potential small molecule-binding sites in proteins. *Journal of Molecular Graphics and Modelling*, 15(6):359–363, December 1997.

[35] P. G. Higgs and T. Attwood. *Bioinformatics and Molecular Evolution*. Blackwell Publishing Limited, January 2005.

[36] L. Holm and J. Park. Dalilite workbench for protein structure comparison. *Bioinformatics*, 16(6):566–567, June 2000.

[37] B. Huang and M. Schroeder. LIGSITEcsc: Predicting ligand binding sites using

the Connolly surface and degree of conservation. *BMC Structural Biology*, 6:19+, September 2006.

[38] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M. Pocock, P. Li, and T. Oinn. Taverna: a tool for building and running workflows of services. *Nucleic Acids Research*, 34(Web Server issue):729–732, July 2006.

[39] M. Jambon, O. Andrieu, C. Combet, G. Deleage, F. Delfaud, and C. Geourjon. The SuMo server: 3D search for protein functional sites. *Bioinformatics*, 21(20):3929–3930, October 2005.

[40] M. Jambon, A. Imberty, G. Deléage, and C. Geourjon. A new bioinformatic approach to detect common 3D sites in protein structures. *Proteins*, 52(2):137–145, August 2003.

[41] D. E. Krane and M. L. Raymer. *Fundamental Concepts of Bioinformatics*. Benjamin Cummings, September 2002.

[42] A. Labarga, F. Valentin, M. Anderson, and R. Lopez. Web Services at the European Bioinformatics Institute. *Nucl. Acids Res.*, 35(Web Server issue), June 2007.

[43] M. Landau, I. Mayrose, Y. Rosenberg, F. Glaser, E. Martz, T. Pupko, and N. Ben-Tal. ConSurf 2005: the projection of evolutionary conservation scores of residues on protein structures. *Nucleic Acids Res*, 33(Web Server issue), July 2005.

[44] M. A. Larkin, G. Blackshields, N. P. Brown, R. Chenna, P. A. Mcgettigan, H. Mcwilliam, F. Valentin, I. M. Wallace, A. Wilm, R. Lopez, J. D. Thompson, T. J. Gibson, and D. G. Higgins. Clustal w and clustal x version 2.0. *Bioinformatics*, 23(21):2947–2948, November 2007.

[45] A. T. Laurie and R. M. Jackson. Q-SiteFinder: an energy-based method for the prediction of protein-ligand binding sites. *Bioinformatics*, 21(9):1908–1916, May 2005.

[46] M. P. Liang, D. R. Banatao, T. E. Klein, D. L. Brutlag, and R. B. Altman. WebFEATURE: An interactive web tool for identifying and visualizing functional sites on macromolecular structures. *Nucleic Acids Research*, 31 (13):3324–3327, 2003.

[47] R. Lopez. 2Can Support Portal, http://www.ebi.ac.uk/2can/home.html.

[48] D. Lupyan, A. Leo-Macias, and A. R. R. Ortiz. A new progressive-iterative algorithm for multiple structure alignment. *Bioinformatics*, June 2005.

[49] M. Malawski, T. Bartynski, and M. Bubak. Invocation of operations from script-based Grid applications. *Future Generation Computer Systems*, May 2009.

[50] M. Malawski, T. Gubala, and M.Bubak. Wirtualne Laboratorium dla chorob zakaznych - ViroLab. rozdzial skryptu, red I. Roterman-Konieczna, (w przygotowaniu).

[51] M. Malawski, D. Kurzyniec, and V. Sunderam. MOCCA - towards a distributed CCA framework for metacomputing. In *Proceedings of 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Joint Workshop on High-Performance Grid Computing and High-Level Parallel Programming Models - HIPS-HPGC, April 4-8, 2005, Denver, Colorado, USA*, page 174a. IEEE Computer Society Press, 2005.

[52] J. L. Moreland, A. Gramada, O. V. Buzko, Q. Zhang, and P. E. Bourne. The Molecular Biology Toolkit (MBT): a modular platform for developing molecular visualization applications. *BMC bioinformatics*, 6, 2005.

[53] A. G. Murzin, S. E. Brenner, T. Hubbard, and C. Chothia. SCOP: a structural classification of proteins database for the investigation of sequences and structures. *Journal of molecular biology*, 247(4):536–540, April 1995.

[54] C. Notredame, D. G. Higgins, and J. Heringa. T-coffee: A novel method for fast and accurate multiple sequence alignment. *Journal of molecular biology*, 302(1):205–217, September 2000.

[55] T. Oinn, M. Greenwood, M. Addis, N. Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin, P. Li, P. Lord, M. Pocock, M. Senger, R. Stevens, A. Wipat, and C. Wroe. Taverna: lessons in creating a workflow environment for the life sciences. *Concurrency and Computation: Practice and Experience*, 18(10):1067–1100, August 2006.

[56] A. R. Ortiz, C. E. Strauss, and O. Olmea. Mammoth (matching molecular models obtained from theory): an automated method for model comparison. *Protein science : a publication of the Protein Society*, 11(11):2606–2621, November 2002.

[57] K. Prymula and I. Roterman. Catalytic residues in hydrolases; analysis of tools destined to ligand-binding site prediction. 2009.

[58] B. Rost and J. Liu. The PredictProtein server. *Nucleic Acids Res*, 31(13):3300–3304, July 2003.

[59] I. Roterman. The geometrical analysis of peptide backbone structure and its local deformations. *Biochimie*, 77 (3):204–216, 1995.

[60] I. Roterman. Modeling the optimal simulation path in the peptide chain folding–studies based on geometry if alanine heptapeptide. *Journal of Theoretical Biology*, 177 (3):283–288, 1995.

[61] I. Roterman, M. Malawski, and T. Jadczyk. Conservative structural element in proteins engaged in immunological signal transduction.

[62] 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.

[63] M. Senger, P. Rice, A. Bleasby, T. Oinn, and M. Uludag. Soaplab2: more reliable Sesame door to bioinformatics programs.

[64] M. Shatsky, R. Nussinov, and H. J. Wolfson. A method for simultaneous alignment of multiple protein structures. *Proteins*, 56(1):143–156, July 2004.

[65] A. Silberschatz, P. B. Galvin, and G. Gagne. *Podstawy Systemow Operacyjnych*. Wydawnictwa Naukowo-Techniczne, 2005.

[66] J. D. Thompson, D. G. Higgins, and T. J. Gibson. Clustal w: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Res*, 22(22):4673–4680, November 1994.

[67] A. M. Waterhouse, J. B. Procter, D. M. A. Martin, M. Clamp, and G. J. Barton. Jalview Version 2–a multiple sequence alignment editor and analysis workbench. *Bioinformatics*, 25(9):1189–1191, May 2009.

[68] I. H. Witten and E. Frank. *Data mining : practical machine learning tools and techniques*. Morgan Kaufmann series in data management systems. Morgan Kaufman, June 2005.

Appendix A

Glossary

The Glossary explains terms used in this thesis, to make bioinformatic definitions intelligible and unequivocal. This Glossary is based on Glossaries from [35] and [41].

A Adenine. One of two purines that are used as nitrogenous bases.

activator Any substance that increases the velocity if an enzyme-catalyzed reaction

active site The region on the three-dimensional surface of a protein where catalysis occurs.

alignment A pairing pf two homologous nucleotide or protein sequences for the purpose of identifying the location of accumulated changes since they last shared a common ancestor.

alleles Different versions of any given gene within a species of organism.

alpha carbon The central carbon in an amino acid to which side chains (R-groups) are bound.

anti-parallel Showing opposite orientation; in the case of double-stranded DNA, this means that if one strand is 5' to 3' orientation, its complementary strand will be in the opposite, 3' to 5' orientation.

backbone (of an amino acid) Consist of amide, an alpha carbon, and a carboxylic acid, or carboxylate group.

base pair (1) The interaction between purines and pyrimidines in double-stranded DNA. (2) The smallest unit of measure of double-stranded DNA length.

C Cytosine. One of two pyrimidines that are used as nitrogenous bases in DNA and RNA molecules.

cDNA Complementary DNA. DNA synthesized from an RNA template by a reverse transcriptase enzyme.

central dogma of molecular biology Process by which information is extracted from nucleotide sequence of a gene and then used to make a protein.

chromosome In prokaryotes, the DNA molecule containing a cell's genome. In eukaryotes, a linear DNA molecule complexed with proteins that contains a large amount of genetic information.

codon Group of three nucleotides in an RNA copy of the coding portion of a gene, corresponding to a specific amino acid.

consensus sequence A sequence that represents the most common nucleotide or amino acid at each position in two or more homologous sequences.

deleterious mutation A mutation that has an adverse effect on the fitness of an organism.

deoxyribonucleic acid (DNA) A usually double-stranded biopolymer of linked nucleotides in which the sugar residue is deoxyribose. The molecular basis of heredity.

enzyme A biological catalyst (usually a protein) that causes a specific chemical reaction to proceed more quickly by lowering its activation energy.

eukaryote An organism whose cells contain complex structures enclosed within membranes. Almost all species of large organisms are eukaryotes, including animals, plants and fungi.

exon A nucleic acid sequence that is represented in the mature form of an RNA molecule after either portions of a precursor RNA (introns) have been removed by cis-splicing or by two or more precursor RNA molecules have been ligated by trans-splicing.

family Consist of protein that are more than 50% identical in amino acid sequence across their entire length.

G Guanine. One of two purines that are used as a nitrogenous base.

gap penalty A reduction in the score for an alignment that is invoked to minimize the introduction of gaps.

gaps A dash or series of dashes introduced to an alignment to reflect the occurence of an indel in one of two aligned sequences sincte they last shared a common ancestor.

gene A specific sequence of nucleotides in DNA or RNA that is essential for a specific function; the functional unit of inheritance controlling the transmission and expression of one or more traits.

gene expression Process of using the information stored in DNA to make an RNA molecule and then a corresponding protein.

genome The sum total of an organism's genetic material.

genotype All or part of the genetic constitution of an individual or group.

global alignment A sequence alignment method that provides a score for aligning two sequences in their entirety.

homologs Sequences that share a common ancestor.

hydrophilic Easily dissolved in a watery solution; literally, "water friendly".

hydrophobic Having limited interaction with water molecules; literally, "afraid of water".

indel event An insertion/deletion event.
intron A DNA region within a gene that is not translated into protein.
length penalty Used by sequence alignmentalgorithms to penalize the introduction of long gaps.
local alignment A sequence alignment method that searches for subsequences that align well.
locus A fixed position on a chromosome.
microarray An ordered grid of DNA probes fixed at known positions on a solid substrate.
multiple sequence alignment An alignment of three or more homologous sequences.
mutation Change in a nucleotide sequence that occurs due to mistake in DNA replication or repair processes. Strictly, changes prior to passage through the filter of selection.
native structure Unique structure into which a particular protein is usually folded within a living cell.
natural selection Differential success between individuals in passing on genes to subsequent generations due to differencies in fitness; leads to changes in allele frequencies (evolution).
neutral mutation A mutation that has no effect on the fitness of an organism.
nucleic acid A macromolecule composed of chains of monomeric nucleotides.
nucleotides Molecules that, when joined together, make up the structural units of RNA and DNA.
oligonucleotide A short nucleic acid polymer, typically with twenty or fewer bases.
orthologs Sequences that share similarity because of a speciation event that allowed them to evolve independently from an ancestral sequence.
paralogs Sequences that share similarity because they are descendants of a duplicated ancestral gene.
peptide A chain of several amino acids.
peptide bond The covalent chemical bond between carbon and nitrogen in a peptide linkage.
pharmacogenomics Field that uses information about an individual's genetic makeup to maximize the efficacy of treatments, while at the same time minimizing the unwanted side effects.
phenotype The visible properties of an organism that are produced by interaction of its genotype and environment
phylogenetic tree A graphical representation of the evolutionary relationship among three or more genes or organisms.
point accepted mutation A mutation that has been "accepted" by natural selection in the sense that organisms bearing the mutation have survived.
polynucleotide A polymeric chain of nucleotides; DNA or RNA molecules.
polypeptide A polymeric chain of amino acids; protein.
primary structure Sequence in which the various amin acids are assembled into a protein.

probe A piece of labeled DNA or RNA or an anti-body that can specifically interact with a molecule of interest.

prokaryotes Are a group of organisms that lack a cell nucleus (= karyon), or any other membrane-bound organelles.

promoter sequence Sequences that are recognized by RNA polymerases as being associated with a gene.

protein backbone The non-side-chain atoms in a polypeptide chain.

proteome The sum total of an organism's proteins.

purine Nucleotides whose nitrogenous bases have a two-ring structure; usually guanine and adenine.

pyrimidine Nucleotides whose nitrogenous bases have a one-ring structure; usually cytosine, thymine and uracil.

quaternary structure The intermolecular interactions that occur when multiple polypeptides associate; overall structure formed by interacting proteins.

reading frame Linear sequence of codons in a protein-coding gene starting with the start codon and ending with a stop codon.

residue The portion of an amino acid that remains as a part of a polypeptide chain. In the context of a peptide or protein, amino acids are generally refered to as residues.

reverse transcriptase A special enzyme used to convert RNA to DNA.

ribosome Complex of proteins and rRNA thar are responsible for catalyzing translation.

ribozyme RNA molecules that are capable of catalyzing specific chemical reactions such as self-cleavage.

RNA polymerase Enzyme responsible for transcription; converts the information in DNA molecules into RNA molecules.

scoring matrix Matrix used to score each nongap position in the alignment.

secondary structure Structural features such as alpha helices and beta sheets of a protein that arise from primary structure.

sequence (1) The linear order of nucleotides in a DNA or RNA molecule or the order of amino acids in a protein. (2) The act of determining the linear order of nucleotides or amino acids in a molecule.

side chain A short chain or group of atoms attached to the central carbon of an amino acid that confers a distinctive chemistry.

start codon Triplet codon (specifically, AUG) at which both prokaryotic and eukaryotic ribosomes begin to translate an mRNA.

stop codon One of three codons (specifically, UGA, UAG and UAA) that does not instruct ribosome to insert a specific amino acid an, thereby, causes translation of an mRNA to stop.

superfamily Groups of protein families that are related by detecTable levels of sequence similarity that are reflective of an ancient evolutionary relationship.

tetriary structure The overal three-dimensional shape of a folded polypeptide chain.

topology The topographical features of a molecule; its configuration.

transcription The first step in the process of gene expression; making an RNA copy of a gene.

transcriptome The complete set of an organism's RNA sequences.

transition Mutation in which a purine (A or G) is replaced with another purine or in which pyrimidine (C or T) is replaced by another pyrimidine.

translation Process of converting the information from the nucleotide sequences in RNA to the amino acid sequences that make a protein.

transversion Mutation in which a purine (A or G) is replaced with pyrimidine (C or T) or vice versa.

triplet code A set of three nucleotides that can be used to specify a particular amino acid during translation by ribosomes.

Appendix B

The experiments

The complete codes of the main virtual laboratory experiments created in the scope of this thesis are listed in this Appendix.

B.1. ProteinStructureAndSequenceComparison

B.1.1. Main script

```
puts "Protein Sequence and Structure Comparison Experiment"

#------------ 1. Require section ----------------------------------------------
#Grid objects usage
require "cyfronet/gridspace/goi/core/g_obj"
#Local classes usage
require 'ProteinUtils'
require 'GnuPlotUtils'
require 'CSVUtils'
#Input data - protein families
require 'InputData'
#Handling and storing result files
require 'results_handling'
require 'base64'#Decoding ClustalW output
#------------ End of Require section ------------------------------------------

#------------ 2. Global Variables and Constants section -----------------------
Email = "Setup@YourEmail.org"
#------------ End of Global Variables and Constants section -------------------

#------------ 3. Grid Objects and Local Objects section -----------------------
puts "Creating Gems"
#Pdb gem to retrieve sequence from pdbid code
  pdb = GObj.create('org.pdb.Pdb')
#EarlyFolding gem
  folding = GObj.create('cyfronet.gridspace.gem.EarlyFolding')
  folding.soap.streamhandler.client.receive_timeout = 60000
#Dbfetch
  dbfetch = GObj.create('uk.ac.ebi.Dbfetch')
```

```
#ClustalW
  clustalw = GObj.create('uk.ac.ebi.ClustalW')
#ClustalWUtils
  clw_utils = GObj.create('cyfronet.gridspace.gem.structure_comp.ClustalWUtils')
#Mammoth
  mammoth = GObj.create('cyfronet.gridspace.gem.structure_comp.Mammoth')
  mammoth.soap.streamhandler.client.receive_timeout = 60000
#Gnuplot
  gnuplot = GObj.create('cyfronet.gridspace.gem.GnuPlot')
# Local objects
  pdb_utils = PdbUtils.new
  csv_utils = CSVUtils.new
  gpl_utils = GnuPlotUtils.new
#------------ End of Grid Objects and Local Objects section --------------------

#------------ 4. Data section --------------------------------------------------
@sequences = {}
@structural_codes = {}
@structures = {}
@mm_structures = {}
@fold_structures = {}
#------------ End of Data section ----------------------------------------------

# 5. Select Input protein family (definitions in InputData file)
@proteins_map = @all
puts "Selected family:  #{@family_names[@proteins_map]}"

#------------ 6. Collecting Data section ---------------------------------------
puts "Collecting input data"
for pdbid in @proteins_map.keys
  # 7. Full pdb file - reading from disk or if it's not present - downloading from database
  puts 'Getting PDB: '+ pdbid
  structure = get_full_pdb_structure(pdbid)
  if structure == nil
    puts 'Downloading PDB: '+ pdbid
    structure = dbfetch.fetchData("PDB:#{pdbid}", 'pdb', 'raw')
    write_full_pdb_structure(pdbid, structure)
  end

  # 8. selecting informations for all chains for protein
  for chain in @proteins_map[pdbid]
    key = "#{pdbid}_#{chain}"

    # 9. Get only atoms from appropriate chain
    chain_structure = pdb_utils.filter_pdb_for_chain(structure, chain)
    @structures[key] = chain_structure

    # 10. Get Aminoacid sequence from pdb file. It is also possible to download sequence from database with pdb
gem
    sequence = pdb_utils.get_aminoacid_sequence_from_file(chain_structure, chain)
    puts 'Sequence for '+ key + ' obtained from PDB file:'
    puts sequence
    @sequences[key] = sequence

    # 11. Perform step back stage, get structural code
    code = get_sc_sequence(key)
    if code == nil
      code = folding.get_structural_code(chain_structure)
      write_sc_sequence(key, code)
    end
    puts "Structural code for protein #{key} (Step-Back Unfolding path):"
    puts code
    @structural_codes[key] = code

    # 12. Prepare structure for multiple structure alignment in Mammoth
    #(it is not necessary but it is optimization step for not passing too much data to gem)
```

```
    mm_structure = get_filtered_pdb_structure(key)
    if mm_structure == nil
      mm_structure = pdb_utils.filter_pdb_for_ca_atoms(chain_structure)
      write_filtered_pdb_structure(key, mm_structure)
    end
    @mm_structures[key] = mm_structure
  end
end
#------------ End of Collecting Data section ----------------------------------

#------------ 13. Aminoacids codes alignment section --------------------------
# Create input data
clustal_aa_sequence = ''
for key in @sequences.keys.sort
  clustal_aa_sequence += ">"+ key + "\n"
  clustal_aa_sequence += @sequences[key] + "\n"
end
puts 'ClustalW Amino-Acid sequence:  '
puts clustal_aa_sequence
write_alignment_input('aa', clustal_aa_sequence)

# 14. Perform ClustalW analysis for AA-sequence
#create and set ClustalW parameters
params = {}
  params['email'] = Email
  params['outorder'] = 'input'
  params['matrix'] = 'blosum'
data = {}
  data['type'] = 'sequence'
  data['content'] = clustal_aa_sequence

# 15. Run clustal computations and wait for results
jobid = clustalw.runClustalW(params, [ data ] )
puts 'ClustalW jobid:  '+ jobid

result = clustalw.checkStatus(jobid)
puts 'ClustalW status:  '+ result
  while result=='RUNNING'
    puts "ClustalW - AA Alignment waitign 5 seconds for data"
    sleep 5
    result = clustalw.checkStatus(jobid)
  end
  puts 'Downloading ClustalW AA alignment results'

# 16. Decoding results
aa_aligment_result = Base64.decode64(clustalw.poll(jobid, 'toolaln'))
write_alignment_output('aa', aa_aligment_result)
#------------ End of Aminoacids codes alignment section ------------------------

#------------ 17. Structural codes alignment section --------------------------
# Create input data
clustal_sc_sequence = ''
for key in @structural_codes.keys.sort
  clustal_sc_sequence += ">"+ key + "\n"
  clustal_sc_sequence += @structural_codes[key]
end
puts 'ClustalW Structural codes sequence:  '
puts clustal_sc_sequence
write_alignment_input('sc', clustal_sc_sequence)

# 18. Perform ClustalW analysis for Structural Codes - sequence
#create and set ClustalW parameters
params = {}
  params['email'] = Email
  params['outorder'] = 'input'
  params['matrix'] = 'id'#it is identity matrix, different than used in AA alignment
```

```
data = {}
  data['type'] = 'sequence'
  data['content'] = clustal_sc_sequence

# 19. Run clustal computations and wait for results
jobid = clustalw.runClustalW(params, [ data ] )
puts 'ClustalW jobid:  '+ jobid
result = clustalw.checkStatus(jobid)
puts 'ClustalW status:  '+ result
while result=='RUNNING'
  puts "ClustalW - Structural Codes Alignment waitign 5 sec for data"
  sleep 5
  result = clustalw.checkStatus(jobid)
end
puts 'Downloading ClustalW SC alignment results'
structural_result = Base64.decode64(clustalw.poll(jobid, 'toolaln'))
write_alignment_output('sc', structural_result)
#------------ End of Structural codes alignment section ------------------------

#------------ 20. Crystal Structures alignment section -------------------------
#21. Create Mammoth input
mm_input = {}
for key in @mm_structures.keys.sort
  key.match(/(\w+)_(\w)/)
  chain = $2
  mm_input[key] = chain
end

#21. Run Mammoth computations
puts 'Performing Crystal structure alignment'
mm_result = mammoth.compare_structures( @mm_structures, mm_input, nil)

puts 'Downloading Crystal structure alignment results'
mm_alignment_result = mm_result['aln']
write_alignment_output('mm', mm_alignment_result)
#------------ End of Crystal Structures alignment section ----------------------

#------------ 22. Printing alignment results section ---------------------------
puts
puts 'Alignment results:'
puts
puts 'Aminoacids Alignment:'
puts aa_aligment_result
puts
puts 'Structural codes alignment:'
puts structural_result
puts
puts 'Crystal Structure alignment:'
puts mm_alignment_result
#------------ End of Printing alignment results section ------------------------

#------------ 23. Calculate W Score section ------------------------------------
# 24. Number of codes used in AA and SC alignments definition
AMINOACIDS = 21
STRUCTURAL = 8
#averaging window parameter
averaging_window_size = 5

# 25. Completing sequences (joining parts in one sequence for each protein) and counting W score
puts "Analyzing Aminoacids alignment"
aa_hash = clw_utils.get_complete_sequences(aa_aligment_result)
aa_score = clw_utils.get_w_score_for_sequences(aa_hash.values, averaging_window_size, AMINOACIDS)
puts "Analyzing Structural codes alignment"
sc_hash = clw_utils.get_complete_sequences(structural_result)
sc_score = clw_utils.get_w_score_for_sequences(sc_hash.values, averaging_window_size, STRUCTURAL)
puts "Analyzing Crystal structure alignment"
```

```
mm_hash = clw_utils.get_complete_sequences(mm_alignment_result)
mm_score = clw_utils.get_w_score_for_sequences(mm_hash.values, averaging_window_size, AMINOACIDS) #it
also uses AA representations
#------------ End of Calculate W Score section ---------------------------------

#------------ 26. Plotting W Score section -------------------------------------
#AA alignment
aa_input = gpl_utils.get_gnuplot_input(aa_score)
#Structural code alignment
sc_input = gpl_utils.get_gnuplot_input(sc_score)
#Crystal Structure alignment
mm_input = gpl_utils.get_gnuplot_input(mm_score)

# 27. Plot each type of data
#Reading gnuplot script
up = GS_UserProxyFactory.create_user_proxy()
script = up.get_file("script.gpl")
#Input data and data names
inputs = [aa_input, sc_input, mm_input]
names = ['Amino Acids', 'Structural Codes', 'Structures']
#Plotting and downloading plots
w_score_pictures = {}
for i in 0 ...  inputs.length
   script1 = script.gsub("NAME", names[i]) #print data name on plot
   puts "Plotting #{names[i]} W score"
   url = gnuplot.plot(script1, inputs[i])
   puts "W-score plot for #{names[i]} alignment is available at:  #{url}"
   write_picture(names[i], @proteins_map, url) #downloading picture
   w_score_pictures[names[i]] = url
  end
#------------ End of Plotting W Score section ----------------------------------

#------------ 28. Creating proteins W profiles section -------------------------
aa_profiles = clw_utils.get_w_profiles_for_proteins(aa_score, aa_hash)
sc_profiles = clw_utils.get_w_profiles_for_proteins(sc_score, sc_hash)
mm_profiles = clw_utils.get_w_profiles_for_proteins(mm_score, mm_hash)
#------------ End of Creating proteins W profiles section ----------------------

#------------ 29. Creating Final results section -------------------------------
proteins_pictures = {}
#For each protein do
for protein in aa_profiles.keys
  protein.match(/(\w+)_(\w)/)
  protein_name = $1
  protein_chain = $2
  samples_count = aa_profiles[protein].length

  # 30. Adjusting Gnuplot script for protein - Name, Chain and Residues number (X Axis length)
  script = up.get_file("multiplot.plt")
  script.gsub!("SAMPLES_COUNT", samples_count.to_s)
  script.gsub!("PROTEIN_NAME", protein_name)
  script.gsub!("PROTEIN_CHAIN", protein_chain)

  # 31. Create Multiplot inputs
  inputs = [
    gpl_utils.create_multiplot_input(aa_profiles[protein]),
    gpl_utils.create_multiplot_input(sc_profiles[protein]),
    gpl_utils.create_multiplot_input(mm_profiles[protein])
  ]

  # 32. Plotting multiplot protein W profiles
  url = gnuplot.multiplot(script, inputs)
  puts "Result for protein #{protein} are available at #{url}"
  write_picture(protein, @proteins_map, url)

  # 33. Creating CSV file with data for appropriate protein
```

```
  csv_file = csv_utils.create_csv_file(protein, @structures[protein], aa_profiles[protein],
 sc_profiles[protein], mm_profiles[protein], ?; , ',')
  write_csv(protein, @proteins_map, csv_file)

  # 34. Updating pdb file with amino acids (temperature - beta field) and structural codes (occupancy field) w
_score
  pdb1 = pdb_utils.put_w_scores_into_pdb_file(@structures[protein], aa_profiles[protein],
sc_profiles[protein], 0, 1)
  write_modified_pdb(protein, '_aa_sc', @proteins_map, pdb1)

  # 35. Updating pdb file with crystal structure (temperature - beta field) w _score
  pdb2 = pdb_utils.put_w_score_into_pdb_file(@structures[protein], mm_profiles[protein])
  write_modified_pdb(protein, '_td', @proteins_map, pdb2)
  proteins_pictures[protein] = url
end

  # 36. Create HTML file with collected results
  write_html(@proteins_map, w_score_pictures, proteins_pictures)
#------------ End of Creating final results section ----------------------------

puts "Protein Sequence and Structure Comparison Experiment FINISHED!"
```

B.1.2. ProteinUtils.rb

```
class PdbUtils

  PDB_FILE_CA_POSITION = 13
  PDB_FILE_RES_NAME_POSITION = 17
  PDB_FILE_CHAIN_POSITION = 21

  PDB_FILE_RES_POSITION = 22
  PDB_FILE_RES_LEN = 5

  PDB_FILE_OCCU_POSITION = 54
  PDB_FILE_TEMP_POSITION = 60
  CA = 'CA'

  def initialize
    @aa_name = {
      'A'=> 'ALA', 'C'=> 'CYS', 'D'=> 'ASP', 'E'=> 'GLU',
      'F'=> 'PHE', 'G'=> 'GLY', 'H'=> 'HIS', 'I'=> 'ILE',
      'K'=> 'LYS', 'L'=> 'LEU', 'M'=> 'MET', 'N'=> 'ASN',
      'P'=> 'PRO', 'Q'=> 'GLN', 'R'=> 'ARG', 'S'=> 'SER',
      'T'=> 'THR', 'V'=> 'VAL', 'W'=> 'TRP', 'Y'=> 'TYR'
    }

    @name_aa = @aa_name.invert
  end

  def filter_pdb_for_chain(pdb_file, chain)
   file_lines = pdb_file.split("\n")
   filtered = ''
   last_line = ''
   id = 0
   for i in 0 ...  file_lines.length
    if file_lines[i].match(/^ATOM/)
     if file_lines[i][PDB_FILE_CHAIN_POSITION, 1] == chain
      id += 1
      last_line = file_lines[i]
      last_line[6, 5] = '%5d'% id
      filtered += last_line
      filtered += "\n"
        end
      end
    end
```

```
    id += 1
    ter_line = "TER   "+ "%5d"% id + "      "+ last_line[17, 10] + (" "* 53) + "\n"
    filtered += ter_line
     return filtered
  end

def filter_pdb_for_ca_atoms(pdb_file)
  file_lines = pdb_file.split("\n")
  filtered = ''
  last_line = ''
  id = 0
  for i in 0 ...  file_lines.length
   if file_lines[i].match(/^ATOM/)
    if file_lines[i][PDB_FILE_CA_POSITION, 2] == CA
     id += 1
     last_line = file_lines[i]
     last_line[6, 5] = '%5d'% id
     filtered += last_line
     filtered += "\n"
        end
      end
    end

    id += 1
    ter_line = "TER   "+ "%5d"% id + "      "+ last_line[17, 10] + (" "* 53) + "\n"
    filtered += ter_line
    return filtered
  end

  def get_aminoacid_sequence_from_file(pdb_file, chain)
    file_lines = pdb_file.split("\n")
    res_id = ''
    residues = ''
    for i in 0 ...  file_lines.length
      if file_lines[i].match(/^ATOM/) and \
        file_lines[i][PDB_FILE_CHAIN_POSITION, 1] == chain and \
         file_lines[i][PDB_FILE_RES_POSITION, PDB_FILE_RES_LEN] != res_id #new resiude
          res_name = file_lines[i][PDB_FILE_RES_NAME_POSITION, 3]
          res = @name_aa[res_name]
          residues += res
          res_id = file_lines[i][PDB_FILE_RES_POSITION, PDB_FILE_RES_LEN]
      end
    end
    return residues
  end

  def put_w_score_into_pdb_file(pdb_file, w_score, move_w = 0) #use temperature field in pdb file
   return put_w_into_temp_and_occu_fields(pdb_file, nil, w_score, 0, move_w)
  end

  def put_w_scores_into_pdb_file(pdb_file, w_score1, w_score2, move_w1 = 0, move_w2 = 0) #use temper-
ature and occupancy field in pdb file
   return put_w_into_temp_and_occu_fields(pdb_file, w_score2, w_score1, move_w2, move_w1)
  end

  private
def put_w_into_temp_and_occu_fields(pdb_file, occu_w_score, temp_w_score, move_occu = 0, move_temp = 0)
  res_file = ''
  lines = pdb_file.split("\n")
  last_res_id = ''
  ind = -1
  for i in 0 ...  lines.length
   if lines[i].match(/^ATOM/)
    res = lines[i][PDB_FILE_RES_POSITION, PDB_FILE_RES_LEN]
    if res != last_res_id
```

```
     last_res_id = res
     ind += 1
        end
    new_line = lines[i]

        if occu_w_score != nil
         index = ind - move_occu
         w_val = 0.0
         if index >= 0 and index < occu_w_score.length
          w_val = occu_w_score[index]
          w_val *= 1000 #w value is in 0.0 .. 1.0, but we've got abcd.xy format available in file, and it is possible to rescale
          w_val = 999.99 if w_val >= 1000.0
            end
            new_line[PDB_FILE_OCCU_POSITION, 6] = "%6.2f"% w_val.to_s
          end
          if temp_w_score != nil
            index = ind - move_temp
            w_val = 0.0
            if index >= 0 and index < temp_w_score.length
              w_val = temp_w_score[ind]
              w_val *= 1000 #w value is in 0.0 .. 1.0, but we've got abcd.xy format available in file, and it is possible to rescale
              w_val = 999.99 if w_val >= 1000.0
            end
            new_line[PDB_FILE_TEMP_POSITION, 6] = "%6.2f"% w_val.to_s
          end
          res_file += new_line + "\n"
        else
          res_file += lines[i] + "\n"
        end
      end
      return res_file
    end
end
```

B.1.3. GnuPlotUtils.rb

```
class GnuPlotUtils
  def get_gnuplot_input(w_score)
    input = ''
    #first column is residue number, second - w value
    for i in 0 ...  w_score.length
     input << (i).to_s + ' '+ w_score[i].to_s + "\n"
    end
    return input
  end

  def create_multiplot_input(values)
    input = ''
   for i in 1 ..  2
    #for color map values have to be doubled
    #x - residue number
    #y - 1 or 2
    #z - w value
    for j in 1 ..  values.length
     input += "#{j} #{i} #{values[j-1]}\n"
      end
    input += "\n"
    end
    input
  end
end
```

B.1.4. CSVUtils.rb

```
require 'csv'

class CSVUtils
PDB_FILE_RES_POSITION = 22
  PDB_FILE_RES_LEN = 5

def create_csv_file(pdb_name, pdb_file, aa_w_score, sc_w_score, td_w_score, separator = ?, , float_point
= '.')  #only temperature_field is used
  #create resiudes ids array
  res_ids = Array.new
  last_res_id = ''
  lines = pdb_file.split("\n")
  for i in 0 ...  lines.length
   if lines[i].match(/^ATOM/)
    res = lines[i][PDB_FILE_RES_POSITION, PDB_FILE_RES_LEN]
    if res != last_res_id
     res_ids << res
     last_res_id = res
    end
   end
  end
  #check length - res_ids length should be the same as aa_w_score and td_w_score and 2 positions shorter than
sc_w_score
  if res_ids.length != aa_w_score.length or res_ids.length - td_w_score.length > 1 \
   or res_ids.length != sc_w_score.length + 2
   raise "Wrong data length for protein #{pdb_name}!  Residues:  #{res_ids.length},
AA W Score:  #{aa_w_score.length}, SC W Score:  #{sc_w_score.length}, 3D W Score:  #{td_w_score.length}"
  end

  csv_file = ''
  #create protein name
  row = [pdb_name]
  CSV.generate_row(row, row.length, csv_file, separator)
  #create headers
  row = ['residue', 'AA W Score', 'SC W Score', '3D W Score']
  CSV.generate_row(row, row.length, csv_file, separator)
  #create first line
  row = [res_ids[0], aa_w_score[0], nil, td_w_score[0]]
  CSV.generate_row(row, row.length, csv_file, separator)
  #create inner lines
  for i in 1 ...  res_ids.length - 1
   row = [res_ids[i], aa_w_score[i], sc_w_score[i-1], td_w_score[i]]
   CSV.generate_row(row, row.length, csv_file, separator)
  end
  #create last line
  len = res_ids.length - 1
  row = [res_ids[len], aa_w_score[len], nil, td_w_score[len]]
  CSV.generate_row(row, row.length, csv_file, separator)
  if float_point != '.'
    csv_file.gsub!('.', float_point)
  end
  return csv_file
end
end
```

B.1.5. results_handling.rb

```
require 'InputData'
require 'net/http'
#Folders:

RESULTS = '../results/'
```

```
COMPLETE_PDB_PATH = RESULTS + 'pdb_structures/complete_pdb/'
FILTERED_PDB_PATH = RESULTS + 'pdb_structures/filtered_pdb/'

AA_SEQUENCE_PATH = RESULTS + 'sequences/aa_sequence/'
SC_SEQUENCE_PATH = RESULTS + 'sequences/sc_sequence/'

ALIGNMENT_PATH = RESULTS + 'alignments/'

# writting pdb structure files onto disk

def write_full_pdb_structure(key, structure)
  write_structure(key, structure, COMPLETE_PDB_PATH)
end

def write_filtered_pdb_structure(key, structure)
  write_structure(key, structure, FILTERED_PDB_PATH)
end

def write_structure(key, structure, dir)
  filename = key.gsub(':', '_') + '.pdb'
  filepath = dir + filename
  file = File.new(filepath, "w")
  file.write(structure)
  file.close
end

def get_full_pdb_structure(key)
  return get_pdb_structure(COMPLETE_PDB_PATH, key)
end

def get_filtered_pdb_structure(key)
  return get_pdb_structure(FILTERED_PDB_PATH, key)
end

def get_pdb_structure(dir, key)
  file_path = dir + key.gsub(':', '_') + '.pdb'
  results = nil
  if File.exist?(file_path)
    results = IO.read(file_path)
  end
  return results
end

# writting sequence files

def write_aa_sequence(key, code)
  filename = key.gsub(':', '_') + '_aa.txt'
  filepath = AA_SEQUENCE_PATH + filename
  file = File.new(filepath, "w")
  file.write(code)
  file.close
end

def write_sc_sequence(key, code)
  filename = key.gsub(':', '_') + '_sc.txt'
  filepath = SC_SEQUENCE_PATH + filename
  file = File.new(filepath, "w")
  file.write(code)
  file.close
end

def get_sc_sequence(key)
  filename = key.gsub(':', '_') + '_sc.txt'
  filepath = SC_SEQUENCE_PATH + filename
  results = nil
  if File.exist?(filepath)
```

```
    results = IO.read(filepath)
  end
  return results
end

# alignments
def write_alignment_input(type, input)
  filename = type + '_align_input.txt'
  filepath = ALIGNMENT_PATH + filename
  file = File.new(filepath, "w")
  file.write(input)
  file.close
end

def write_alignment_output(type, output)
  filename = type + '_align_output.txt'
  filepath = ALIGNMENT_PATH + filename
  file = File.new(filepath, "w")
  file.write(output)
  file.close
end

@pictures_dir = {
  @vcam => RESULTS + 'pictures/vcam/',
  @vcam_icam => RESULTS + 'pictures/vcam_icam/',
  @igg => RESULTS + 'pictures/igg/',
  @all => RESULTS + 'pictures/all/',
  @igg_vcam => RESULTS + 'pictures/igg_vcam/',
  @igg_icam => RESULTS + 'pictures/igg_icam/',
  @temp_doubled => RESULTS + 'pictures/old/',
}
# pictures
def write_picture(name, proteins_map, url)
  dir = @pictures_dir[proteins_map]
  path = dir + name.gsub(' ', '_').gsub(':', '_') + '.png'

  address = URI.parse(url)
  @http = Net::HTTP.new(address.host, address.port)
  resp = @http.get2(address.path)

  png = File.new(path, "wb")
  png.write(resp.body)
  png.close
end

def write_csv(name, proteins_map, csv_file)
  dir = @pictures_dir[proteins_map]
  path = dir + name.gsub(':', '_') + '.csv'

  csv = File.new(path, "w")
  csv.write(csv_file)
  csv.close
end

def  write_modified_pdb(protein, suffix, proteins_map, pdb)
  dir = @pictures_dir[proteins_map]
  path = dir + protein.gsub(':', '_') + suffix + '.pdb'

  file = File.new(path, "w")
  file.write(pdb)
  file.close
end

@texts = {
  'AminoAcids' => 'W Score for Amino Acid alignment sequences (21 codes)',
  'StructuralCodes' => 'W score for Structural codes alignment sequences (8 codes)',
```

```
  'Structures' => 'W Score for 3D Structure alignment (21 codes)'
}
@html_names = {
  @vcam =>  'vcam.html',
  @vcam_icam =>  'vcam_icam.html',
  @igg => 'igg.html',
  @all => 'all.html',
  @igg_vcam => 'igg_vcam.html',
  @igg_icam => 'igg_icam.html',
}
def write_html(proteins_map, w_score_pictures, proteins_pictures)
  body = ''
  body += "<html>\n"
  #write creation time
  body += "<head>\n"
  body += "  <meta http-equiv=\"Creation-Date\" content=\"Time.now.gmtime\" />\n"
  body += "</head>"

  #write body
  body += "<body>\n"

  #write w scores
  for type in @texts.keys
    url = w_score_pictures[type]
    if url != nil
      body += "<div align=\"center\"> \n"
      body += "#{@texts[type]}\n"
      body += "<IMG src=\"#{url}\"> \n"
      body += "</div>\n"
    end
  end
  #write bar

  body += "<hr>\n"

  #write proteins profiles
  body += "<div align=\"center\">\n"
  for protein in proteins_pictures.keys.sort
    body += "<IMG src=\"#{proteins_pictures[protein]}\" alt=\"#{protein} profile\" align=\"left\" border=\"0\">
\n"
  end
  body += "</div>\n"

  #close tags
  body += "</body>\n"
  body += "</html>\n"

  #write_file
  filepath = @pictures_dir[proteins_map] + @html_names[proteins_map]
  file = File.new(filepath, "w")
  file.write(body)
  file.close
end
```

B.2. ProteinPocketsTests

B.2.1. Main script

```
# ProteinPocketsTests.rb
# Experiment Name : ProteinPocketsTests
# Author Email Address : jadczyk@student.agh.edu.pl
# Organization :
# License File : ExperimentLicense.txt
```

```
# ViroLab specific requires
require "cyfronet/gridspace/goi/core/g_obj"
require "cyfronet/gridspace/dac/DACConnectClass.rb"

require "file_utils.rb"
require "service_names.rb"
require "services_utils.rb"
require "tasks_utils.rb"
require "TaskListCreator"

#Directories variables
FILES_DIR = "../pdb/"#select directory with your pdb files
RESULTS_DIR = "../results/"#select directory where results should be store
Use_Separate_Results_Dir = false #if true - separate dir is created in RESULTS_DIR for each analyzed file
  #if false - all results are store in RESULTS_DIR
CreateAllTasks = true #if false - try to search which files left to compute on which Service, if true - all files are
treated as tasks


#Services - uncoment service which You want use
Services = {
  CASTP => GObj.create('cyfronet.gridspace.gem.pocket.CastP'),
  CONSURF => GObj.create('cyfronet.gridspace.gem.pocket.ConSurf'),
  FOD => GObj.create('cyfronet.gridspace.gem.pocket.Fod'),
  PASS => GObj.create('cyfronet.gridspace.gem.pocket.Pass'),
#   LIGSITE => GObj.create('cyfronet.gridspace.gem.pocket.Ligsite_csc'),
#   PFINDER => GObj.create('cyfronet.gridspace.gem.pocket.PocketFinder'),
#   QFINDER => GObj.create('cyfronet.gridspace.gem.pocket.QSiteFinder'),
#   SUMO => GObj.create('cyfronet.gridspace.gem.pocket.SuMo'),
#   WEBFEATURE => GObj.create('cyfronet.gridspace.gem.pocket.WebFeature'),
}

#Additional info
UserEmail = 'setup.your@email.com'

Converter = GObj.create('cyfronet.gridspace.gem.pocket.ProteinResultsConverter')
Converter.soap.streamhandler.client.receive_timeout = 360 #seconds

#reading available files and files paths from direcotry
pdb_files_names = get_pdb_files(FILES_DIR)
pdb_files_paths = get_pdb_files_paths(FILES_DIR, pdb_files_names)
pdb_files_map = {}

for path in pdb_files_paths.keys
  puts path + ' => '+ pdb_files_paths[path]
end

initial_tasks = {}
tasks_left = {}
processing_tasks = {}
log_tasks = {}

if CreateAllTasks
  #initialize maps -> starting tasks (all), processing_tasks (empty), log_tasks
  for service in Services.keys
    initial_tasks[service] = Hash.new
    processing_tasks[service] = Hash.new

    for file_name in pdb_files_paths.keys
     chains = get_chains(service, pdb_files_paths[file_name])
     if is_all_chains_identifier(chains)
      #add initial tasks
      initial_tasks[service][file_name] = Array.new
      initial_tasks[service][file_name] << chains
      #add log info
      id = "#{file_name}:#{chains}"
```

```
      if log_tasks[id] == nil
       log_tasks[id] = Hash.new
      end
      log_tasks[id][service] = ProcessingStatus::NOT_COMPLETED
      else
      #add initial tasks
      initial_tasks[service][file_name] = chains
      #add log info
      for ch in chains
       id = "#{file_name}:#{ch}"
       if log_tasks[id] == nil
        log_tasks[id] = Hash.new
       end
       log_tasks[id][service] = ProcessingStatus::NOT_COMPLETED
      end
      end
    end
  end
else
  task_list = TaskListCreator.new
  initial_tasks, processing_tasks, log_tasks =
   task_list.get_tasks_list(FILES_DIR, RESULTS_DIR, Services.keys, Use_Separate_Results_Dir)
end

#deep copy of initial set of tasks
tasks_left = Marshal::load(Marshal.dump(initial_tasks))

puts 'Processing started!'
begin
  #put initial set of tasks to services
  MaxTasks = 6
  for service in Services.keys
   tasks_todo = get_tasks_todo(service, tasks_left, MaxTasks)
   puts service + ':'+ tasks_todo.join(' ')
   for task in tasks_todo
    options = nil
    if service == CONSURF
     options = {'chain'=> task[1], 'user_email'=> UserEmail}
     end
    taskid = Services[service].get_results_from_file(
     task[0], get_file(task[0], pdb_files_paths, pdb_files_map),
          options, get_output_types(service))
    puts 'TaskID: '+ taskid
    add_processing_task(service, processing_tasks, task, taskid)
    puts processing_tasks[service][id]
    remove_task_todo(service, tasks_left, task)
    end
  end

  while not processing_tasks.empty?
    retrieved = 0
    for service in processing_tasks.keys
      for id in processing_tasks[service].keys
        puts "taksid:"+ processing_tasks[service][id]
        status = Services[service].get_status(processing_tasks[service][id])
        puts "Service:  #{service}, Task:  #{id} - #{ProteinTaskStatus.to_s(status)}"
        if status ==  3 #ProteinTaskStatus::FINISHED
          #get result from service
          retrieved += 1
          puts "Retrieving from service:  #{service} task with ID: #{processing_tasks[service][id]}"
          result = Services[service].get_result(processing_tasks[service][id])
          #handle results
          if result.result_type == ProteinTaskResultType::OK
           puts "Service:  #{service}, Task:  #{id} - Result OK!"
           #add log info
           log_tasks[id][service] = ProcessingStatus::OK
```

```
            file_type = "w"
           file_type = "wb"if service == WEBFEATURE
           id.match(/(\S+):(\S+)/)
            name = $1
           chain = nil
           chain = $2 if service == CONSURF
            #store all result files
            for output in get_output_types(service)
             path = create_path(RESULTS_DIR, name, chain, service, get_result_extension(service, output),
Use_Separate_Results_Dir)
             store_file(path, result.results[output], file_type)
            end

            pocket_info = ''
            write_default = true
            if service== CONSURF
             gradesPE = result.results['gradesPE']
             colors = ['7', '8', '9']
             for color in colors
              conv_options = { 'chain'=> chain, 'color'=> color}
              pocket_info = Converter.get_atoms_and_chains(service, gradesPE, name, conv_options)
              path = create_path(RESULTS_DIR, name, chain + "_#{color}", service, 'common.txt',
Use_Separate_Results_Dir)
              store_file(path, pocket_info, "w")
             end
            write_default = false
            elsif service == LIGSITE
            pocket = result.results['pocket']
            binding = result.results['ligand']
            pdb = get_file(name, pdb_files_paths, pdb_files_map)
            conv_options = { 'pdb'=> pdb, 'binding'=> binding}
            pocket_info = Converter.get_atoms_and_chains(service, pocket, name, conv_options)
            elsif service == PASS
            output = result.results['pass']
            conv_options = { 'pdb'=> get_file(name, pdb_files_paths, pdb_files_map)}
           pocket_info = Converter.get_atoms_and_chains(service, output, name, conv_options)
            elsif service == WEBFEATURE
           tar_file = result.results['significant_hits']
           files = get_webfeature_sig_files(name, tar_file)
           options = [95, 99, 100]
           for opt in options
            conv_options = { 'opt'=> opt }
            pocket_info = Converter.get_atoms_and_chains(service, files, name, conv_options)
            path = create_path(RESULTS_DIR, name, nil, "#{opt}_"+ service, 'common.txt',
Use_Separate_Results_Dir)
            store_file(path, pocket_info, "w")
           end
          write_default = false
            else
          #default - for services: CASTP, FOD, PFINDER, QFINDER, SUMO
          output = result.results[get_output_types(service)[0]]
          conv_options = nil
          pocket_info = Converter.get_atoms_and_chains(service, output, name, conv_options)
            end

        if write_default
         path = create_path(RESULTS_DIR, name, nil, service, 'common.txt', Use_Separate_Results_Dir)
         store_file(path, pocket_info, "w")
            end

        puts "Service:  #{service}, Task:  #{id} - Results Stored!"
          else #ERROR
            puts "Service:  #{service}, Task:  #{id} - Error!"
            #add log info
            log_tasks[id][service] = ProcessingStatus::ERROR
            id.match(/(\S+):(\S+)/)
```

```
            name = $1
            chain = nil
            chain = $2 if service == CONSURF
            error_info = result.error_info
            path = create_path(RESULTS_DIR, name, chain, service, 'ERROR.txt', Use_Separate_Results_Dir)
            store_file(path, error_info, "w")
          end

          #remove this task from processing tasks map
          remove_processing_task(service, processing_tasks, id)
          #try to send next task to this service
          next_task = get_next_task(service, tasks_left)
          if next_task != nil
            puts "Sending new task:  #{next_task[0]}:#{next_task[1]} to service #{service}"
            options = nil
            if service == CONSURF
             options = {'chain'=> next_task[1], 'user_email'=> UserEmail}
            end
            taskid = Services[service].get_results_from_file(
        next_task[0], get_file(next_task[0], pdb_files_paths, pdb_files_map),
         options, get_output_types(service))
            add_processing_task(service, processing_tasks, next_task, taskid)
            remove_task_todo(service, tasks_left, next_task)
           end
         end
       end
     end

     if retrieved == 0
       puts "No results available - waiting 10 sec."
       sleep(10.0) #wait 10 seconds
     else
       puts "Retrieved:  #{retrieved} files"
     end
   end

rescue Exception => e
  puts 'Error during computations!  '+ e
  puts e.backtrace.join("\n")
ensure
  #write log
  log_file = ''
  for id in log_tasks.keys.sort
   for service in log_tasks[id].keys.sort
    log_file += "#{id} => #{service} => "+ ProcessingStatus.to_s(log_tasks[id][service]) + "\n"
    end
  end
  log_file_path = RESULTS_DIR + File::Separator + 'all.log'
  store_file(log_file_path, log_file, "w")
end

puts 'Experiment finished'
```

B.2.2. file_utils.rb

```
require 'fileutils'

# reading file
def read_file(file_path)
  if(!  File.exist?(file_path))
   raise 'File not found!  '+ file_path.to_s
  end
  file_body = IO.read(file_path)
  return file_body
end
```

```
#saving file
def store_file(file_path, filebody, mode)
  #try to create dir if it doesn't exist
  dirpath = File.dirname(file_path)
  FileUtils.mkdir_p dirpath

  outfile = File.new(file_path, mode)
  outfile.write(filebody)
  outfile.close
end

def create_path(directory, name, chain,  service, ext, use_separate_dir)
  path = ''
  if chain == nil
   path = directory + File::Separator + name + '_'+ service + '.'+ ext
   path = directory + File::Separator + name + File::Separator + name + '_'+ service + '.'+
ext if use_separate_dir
  else
   path = directory + File::Separator + name + '_'+ chain + '_'+ service + '.'+ ext
   path = directory + File::Separator + name + File::Separator + name + '_'+ chain  + '_'+
service + '.'+ ext if use_separate_dir
  end

  return path
end

def create_webfeature_tmp_dir()
path = "tmp_dir_"+ (Time.now.to_f * 10000).to_i.to_s + File::Separator
FileUtils.mkdir_p path
return path
end

#reading pdb files names from dir
def get_pdb_files(dir_path)
  pdb_files = Array.new
  Dir.foreach(dir_path) { |filename|
    if(filename.match(/.*\.pdb\Z/i))
        pdb_files << filename
    end
  }
  return pdb_files
end


def get_pdb_files_paths(dir_path, pdb_files)
pdb_paths = Hash.new
for pdb in pdb_files
  pdb.match(/(.*)\.pdb\Z/i)
  file_key = $1
  pdb_paths[file_key] = File.join(dir_path, pdb)
  end
return pdb_paths
end

def get_file(pdb_file_name, pdb_file_paths, pdb_file_map)
  if(pdb_file_map[pdb_file_name] == nil)
    #read file from disk
     pdb_file_map[pdb_file_name] = read_file(pdb_file_paths[pdb_file_name])
  end
  return pdb_file_map[pdb_file_name]
end
```

B.2.3. service_names.rb

```
CASTP = 'CastP'
CONSURF = 'ConSurf'
FOD = 'Fod'
LIGSITE = 'Ligsite'
PASS = 'Pass'
PFINDER = 'PocketFinder'
QFINDER = 'QSiteFinder'
SUMO = 'SuMo'
WEBFEATURE = 'WebFeature'
```

B.2.4. services_utils.rb

```
require "service_names.rb"
require "file_utils.rb"

ALL_IDENTIFIER = "ALL"

@@outputs = {CASTP => ['poc'], CONSURF => ['gradesPE'], FOD => ['fod'], LIGSITE => ['pocket', 'ligand'],
      PASS => ['pass'], PFINDER => ['pdb'], QFINDER => ['pdb'], SUMO => ['text'],
WEBFEATURE => ['significant_hits'] }

@@ext = {
  CASTP => { 'pdb'=> 'pdb', 'mouth'=> 'mouth', 'mouth_info'=> 'mouthInfo', 'poc'=> 'poc', 'poc_info'=>
'pocInfo'},
  CONSURF => {'gradesPE'=> 'gradesPE', 'blast'=> 'blast', 'fasta'=> 'homol.html', 'alignment'=> 'aln',
'tree'=> 'tree.txt',
    'rasmolisd'=> 'rasmolisd.txt', 'rasmol'=> 'rasmol.txt', 'pdb'=> 'ent', 'consurf'=> 'pipe.pdb'},
  FOD => { 'fod'=> 'dat'},
  LIGSITE => {'pdb'=> 'pdb', 'pocket'=> 'pocket.pdb', 'ligand'=> 'ligand.txt'},
  PASS => { 'pass'=> 'probes.pdb'},
  PFINDER => {'pdb'=> 'pdb'},
  QFINDER => {'pdb'=> 'pdb'},
  SUMO => { 'text'=> 'txt'},
  WEBFEATURE => {'significant_hits'=> 'sig.tar', 'insignificant_hits'=> 'insig.tar',
    'coordinates'=> 'points.tar', 'features'=> 'features.tar'},
}

#used for split task into subtasks for ConSurf analyzer
def get_chains_identifiers(pdb_file)
lines = pdb_file.split("\n")
chains = Array.new
for l in lines
  if l.match(/^ATOM/)
   ch = l[21, 1] #PDB specification - column 22 - chain identifier
   if !  chains.include?(ch)
    chains << ch
      end
    end
  end
return chains
end

def get_chains(service, file_path)
  if service != CONSURF
    return ALL_IDENTIFIER
  end
  return get_chains_identifiers(read_file(file_path))
end

def is_all_chains_identifier(identifier)
  return identifier == ALL_IDENTIFIER
end
```

```
def get_output_types(service)
return @@outputs[service]
end


def get_result_extension(service, result_type)
  return @@ext[service][result_type]
end

def get_webfeature_sig_files(name, tar_file)
  #create tmp dir
  tmp_dirname = create_webfeature_tmp_dir
  #create tmp file (store data)
  tmp_file_path = name + '.tar'
  store_file(tmp_dirname + tmp_file_path, tar_file, "wb")
  #untar file in tmp dir
  res = `tar xf #{tmp_dirname}#{tmp_file_path} -C #{tmp_dirname}`
  #read all files whose match to /zscore/
  sig_dir = tmp_dirname + "sig/"
  files = {} #result
  Dir.foreach(sig_dir) { |file|
   if file.match(/zscores.sort$/)
    body = read_file(sig_dir + file)
    files[file] = body
    end
  }
  #remove whole tmp dir
  `rm -Rf #{tmp_dirname}`
  return files
end

class ProcessingStatus
  OK = 0
  ERROR = 1
  NOT_COMPLETED = 2
  @@StatusMap = { OK => 'OK', ERROR => 'ERROR', NOT_COMPLETED => 'Not completed'}

  def ProcessingStatus.to_s(status)
   @@StatusMap[status]
  end
end

class ProteinTaskStatus
  CREATED = 0
  QUEUED = 1
  PROCESSING = 2
  FINISHED = 3

  def ProteinTaskStatus.to_s(status)
    status_map = { 0 => 'Created', 1 => 'Queued', 2=> 'Processing', 3 => 'Finished'}
    return status_map[status]
  end
end

class ProteinTaskResultType
  OK = 0
  ERROR = 1
  NO_RESULT = -1
end
```

B.2.5. task_utils.rb

```
def get_tasks_todo(service, tasks_left, maxTasks)
  results = Array.new
  service_tasks = tasks_left[service]
```

```
  if service_tasks != nil
    available_tasks = service_tasks.keys
    found = 0
    index = 0
    while (found < maxTasks and index < available_tasks.length)
     tasks_chains = tasks_left[service][available_tasks[index]]
     len = maxTasks - found
     len = tasks_chains.length if tasks_chains.length < (maxTasks - found)
     for i in 0 ...  len
      results << ["#{available_tasks[index]}", "#{tasks_chains[i]}"]
     end
      found += len
      index += 1
    end
  end
  return results
end

def get_next_task(service, tasks_left)
  return get_tasks_todo(service, tasks_left, 1)[0]
end

def remove_task_todo(service, tasks_left, task)
  filename = task[0]
  chainid = task[1]
  tasks_left[service][filename].delete(chainid)
  tasks_left[service].delete_if { |key, array|array.empty?  }
  tasks_left.delete_if { |key, hash|hash.empty?  }
end

def add_processing_task(service, processing_tasks, task, taskid)
  identifier = "#{task[0]}:#{task[1]}"
  if processing_tasks[service] == nil
    processing_tasks[service] = Hash.new
  end
  processing_tasks[service][identifier] = taskid
end

def remove_processing_task(service, processing_tasks, identifier)
  processing_tasks[service].delete(identifier)
  processing_tasks[service].delete_if { |key, hash|hash.empty?  }
  processing_tasks.delete_if { |key, hash|hash.empty?  }
end
```

B.2.6. TaskListCreator.rb

```
require 'service_names'
require 'file_utils'
require 'services_utils'

class TaskListCreator

  def get_tasks_list(inputs_dir, results_dir, services, use_separate_dir)
    pdb_files_names = get_pdb_files(inputs_dir)
    pdb_files_paths = get_pdb_files_paths(inputs_dir, pdb_files_names)
    pdb_files = pdb_files_paths.keys

    initial_tasks = {}
    processing_tasks = {}
    log_tasks = {}

    for name in pdb_files
      dir = results_dir + File::Separator
      dir = results_dir + File::Separator + name  + File::Separator if use_separate_dir
```

```
    for service in services
      chains = get_chains(service, pdb_files_paths[name])
      #get necessary results file names for service (only common files)
      files = get_results_files_for_service(name, service, chains)
      #get error file name for service
      errors = get_error_files_for_service(name, service, chains)
      #check files presence
      chains = get_tasks_left(files, errors, dir, chains)
      #if both file type are not available - add task to compute
      if not chains.nil?
        #add task
        initial_tasks[service] = Hash.new if initial_tasks[service] == nil
        processing_tasks[service] = Hash.new if processing_tasks[service] == nil
        if is_all_chains_identifier(chains)
         #add initial tasks
         initial_tasks[service][name] = Array.new if initial_tasks[service][name] == nil
         initial_tasks[service][name] << chains
         #add log info
         id = "#{name}:#{chains}"
         log_tasks[id] = Hash.new if log_tasks[id] == nil
         log_tasks[id][service] = ProcessingStatus::NOT_COMPLETED
        else
         #add initial tasks
         initial_tasks[service][name] = chains
         #add log info
         for ch in chains
          id = "#{name}:#{ch}"
          if log_tasks[id] == nil
           log_tasks[id] = Hash.new
          end
          log_tasks[id][service] = ProcessingStatus::NOT_COMPLETED
         end
        end
      end
    end
  end
  return initial_tasks, processing_tasks, log_tasks
end

private
def get_results_files_for_service(name, service, chains)
  results = Hash.new
  if service == WEBFEATURE
    chain_res = Array.new
   opts = ['95', '99', '100']
   for opt in opts
    file_name = name + '_'+ opt + "_"+ service + ".common.txt"
    chain_res << file_name
    end
   results[chains] = chain_res
  elsif service == CONSURF
   colors = ['7', '8', '9']
   for chain in chains
    chain_res = Array.new
    for color in colors
     file_name = name + '_'+ chain + "_"+ color + '_'+ service + ".common.txt"
     chain_res << file_name
      end
    results[chain] = chain_res
    end
  else
    file_name = name + '_'+ service + ".common.txt"
    chain_res = [file_name]
    results[chains] = chain_res
  end
  return results
```

```
  end

  def get_error_files_for_service(name, service, chains)
    results = Hash.new
    if service == CONSURF
      for chain in chains
        file_name = name + '_'+ chain + '_'+ service + '.ERROR.txt'
        chain_res = [file_name]
        results[chain] = chain_res
      end
    else
      file_name = name + '_'+ service + '.ERROR.txt'
      chain_res = [file_name]
      results[chains] = chain_res
    end
      return results
  end

  def are_all_files_available(dir, files)
    presence = true
    for file in files
      if not is_file_available(dir, file)
        presence = false
        break
      end
    end
    return presence
  end

  def is_file_available(dir, file)
    path = File::join(dir, file)
    return File::exist?(path)
  end

  def get_tasks_left(files, errors, dir, chains)
    if is_all_chains_identifier(chains)
     if not are_all_files_available(dir, files[chains]) and not are_all_files_available(dir, errors[chains])
      return chains
      else
      return nil
      end
    else
      res_chains = Array.new
      for chain in chains
        if not are_all_files_available(dir, files[chain]) and not are_all_files_available(dir, errors[chain])
          res_chains << chain
        end
      end
      if res_chains.empty?
       return nil
      else
        return res_chains
      end
    end
  end
end
```

Appendix C

Gems API

This Appendix presents the complete API of developed gems. The gems are organized in sections corresponding to the usage scope layers, presented in section 4.2.2.

C.1. Bioinformatic database access gems

Scop
Gem class: `cyfronet.gridspace.gem.bioinfo.data.ScopDb`
Description: Searching SCOP database for protein families.
Technology: Local Gem, JRuby class.
Methods:

- `get_family_pdbids(family_name:String) : String[]`
  search SCOP database on family name; `family_name`: the name of sought family; returns: an array of pdbid codes of proteins that belong to sought family or empty array if family was not found

DbFetch
Gem class: `uk.ac.ebi.Dbfetch`
Description: Retrieve entries from various up-to-date biological databases using entry identifiers or accession numbers
Technology: External Web service
Methods:

- `fetchData(query:String, format:String, style:String) : String[]`
  Fetch an entry in a defined format and style; `query`: the entry identifier in *db:id* format; `format`: the name of the format required; `style`: the name of the style required; returns: an array of strings containing the entry. Generally this will contain only one item which contains the whole entry.
- `getDBFormats(db:String) : String[]`
  Get a list of format names for a given database; `db`: database name to get available formats for; returns: an array of strings containing the format names.
- `getFormatStyles(db:String, format:String) : String[]`
  Get a list of style names available for a given database and format; `db`: database name to get available styles for; `format`: the data format to get available styles for; returns: an array of strings containing the style names.

- `getSupportedDBs() : String[]`
  Get a list of database names usable with WSDbfetch; returns: an array of strings containing the database names.
- `getSupportedFormats() : String[]`
  Get a list of database and format names usable with WSDbfetch; returns: an array of strings containing the database and format names.
- `getSupportedStyles() : String[]`
  Get a list of database and style names usable with WSDbfetch; returns: an array of strings containing the database and style names.

Pdb

Gem class: `org.pdb.Pdb`
Description: The Interface for the RCSB PDB Web service. A number of different functions are provided.
Technology: External Web service.
Methods:

- `getChainLength(structureId:String, chainId:String) : Integer`
  Finds the length of the given chain; `structureId`: a PDB Structure Identifier (aka PDB ID); `chainId`: an author-assigned Chain Identifier; returns: the length of the given chain.
- `getChains(structureId:String) : String[]`
  Finds all the chain identifiers for a given structure; `structureId`: a PDB Structure Identifier (aka PDB ID); returns: an array of the author-assigned chain identifiers for the given structure.
- `getSequenceForStructureAndChain(structureId:String, chainId:String) : String`
  Finds the sequence for the given chain; `structureId`: a PDB Structure Identifier (aka PDB ID); `chainId`: an author-assigned Chain Identifier; returns: a sequence.
- `runXmlQuery(xmlQuery:String) : String[]`
  This is the most important Web service call provided by RCSB-PDB, since it allows to access all of the advanced search functionality via a Web service. Runs any RCSB advanced query. This works by posting the XML representation of an advanced query; `xmlQuery`: the XML representing a RCSB advanced query; returns: an array of PDB IDs.

GeoDataProviderService

Gem class: `cyfronet.gridspace.gem.microarray.GeoDataProviderService`
Description: Retrieving microarray data from NCBI Geo and EBI ArrayExpress databases. Downloading existing datasets and samples as well as creating new datasets is possible.
Technology: Web service. Service was implemented as a *Ruby* class
Methods:

- `create_dataset(samples_ids:String[], db_type:String) : String`
  Create microarray dataset from microarray samples; `samples_ids`: an array of samples identifiers; `db_type`: database identifier; returns: text file with created dataset.
- `create_dataset_obj(samples_ids:String[], db_type:String) : GeoDataSet`
  Create microarray dataset from microarray samples; `samples_ids`: an array of samples identifiers; `db_type`: database identifier; returns: created dataset object.
- `get_dataset(gds_id:String, db_type:String) : String`
  Download microarray dataset from database; `gds_id`: dataset identifier; `db_type`: database identifier; returns: text file with downloaded dataset.
- `get_dataset_obj(gds_id:String, db_type:String) : GeoDataSet`
  Download microarray dataset from database; `gds_id`: dataset identifier; `db_type`: database identifier; returns: downloaded dataset object.
- `get_sample(gsm_id:String, db_type:String) : String`
  Download single microarray sample from database; `gsm_id`: sample identifier; `db_type`: database identifier; returns: text file with downloaded sample.

- `get_sample_obj(gsm_id:String, db_type:String) : GeoSample`
  Download single microarray sample from database; `gsm_id`: sample identifier; `db_type`: database identifier; returns: downloaded sample object.

FormatConverter
Gem class: `cyfronet.gridspace.gem.bioinfo.data.FormatConverter`
Description: Performs conversion between various data formats.
Technology: Web service. Service was implemented as a *Ruby* class
Methods:

- `convert_arff_to_double(arff:String) : Double[],String[]`
  `arff`: data file in ARFF format; returns: an array with data, an array with attributes names.
- `convert_arff_to_geo(arff:String, params:Hash) : GeoDataSet`
  `arff`: data file in ARFF format; `params`: dataset parameters; returns: dataset object.
- `convert_double_to_arff(values:Double[], params:String[]) : String`
  `values`: attributes values; `params`: attributes names; returns: data file in ARFF format.
- `convert_double_to_geo(values:Double[], params:Hash) : GeoDataSet`
  `values`: gene expression levels; `params`: dataset parameters; returns: dataset object.
- `convert_geo_to_arff(gds:GeoDataSet) : String,Hash`
  `gds`: dataset; returns: data file in ARFF format, dataset parameters.
- `convert_geo_to_double(gds:GeoDataSet) : Double[],Hash`
  `gds`: dataset; returns: an array with data, dataset parameters.

C.2. Basic analysis gems

R_gem
Gem class: `cyfronet.gridspace.gem.r2`
Description: Execute scripts that are written in *R* language.
Technology: Web service, service is a binary program wrapper, written in *Ruby*.
Methods:

- `check_R_version() : String`
  Checks the version of the R program installed on a remote host; returns: Result of executing R –version.
- `execute_R_code(code:String, inputs:Hash) : String[]`
  Executes R code given in the code parameter; `code`: script to execution; `inputs`: contains pairs id => file name; returns: an array of URLs with created results

WekaAssociator
Gem class: `cyfronet.gridspace.gem.weka.WekaAssociator`
Description: Data mining tool for discovering association rules in data.
Technology: Mocca component, *Weka* library integration, written in *Java*.
Methods:

- `assignAssociator(associatorPath:String, options:String[])`
  Create an associator object and set options to it; `associatorPath`: complete Java classpath for assigned Associator; `options`: options to set for an associator.
- `createAssociations(dataset:String) : String`
  Create associations rules; `dataset`: instances data; returns: created rules.
- `createAssociationsURL(dataURL:String, dataUser:String) : String`
  Create associations rules from dataset stored in WebDav repository; `dataURL`: an URL of instances; `dataUser`: 'user:password'; returns: created rules.

WekaClassifier
Gem class: `cyfronet.gridspace.gem.weka.WekaClassifier`
Description: Creating, training and testing classifiers. Supervised learning.

Technology: Mocca component, *Weka* library integration, written in *Java*.
Methods:

- `assignClassifier(classifierPath:String, options:String[])`
  Create a classifier object and set options to it; `classifierPath`: complete Java classpath for assigned Classifier; `options`: options to set for a classifier.
- `train(dataset:String, attributeName:String) : long`
  Learn classifier for predict attribute; `dataset`: instances data; `attributeName`: attribute to learn for further predictions; returns: learning time.
- `classify(dataset:String) : String`
  Classify instances for attribute passed in training time; `dataset`: instances data; returns: instances with predicted data.
- `trainURLData(dataURL:String, dataUser:String, attributeName:String) : long`
  Learn classifier. Data provided are stored in WebDav repository; `dataURL`: an URL of instances; `dataUser`: 'user:password'; `attributeName`: attribute to learn for further predictions; returns: learning time.
- `classifyURLData(dataURL:String, dataUser:String, classifiedDataURL:String, classifiedDataUser:String) : String`
  Classify instances which are stored in WebDav repository. Store results also in repository; `dataURL`: an URL of instances; `dataUser`: 'user:password' for reading data; `classifiedDataURL:` address to store classified data; `classifiedDataUser:` 'user:password' for storing data; returns: URL of classified data.

WekaClusterer
Gem class: `cyfronet.gridspace.gem.weka.WekaClusterer`
Description: Using *Weka* clusterers. Unsupervised learning.
Technology: Mocca component, *Weka* library integration, written in *Java*.
Methods:

- `assignClusterer(clustererPath:String, options:String[])`
  Create a clusterer object and set options to it; `clustererPath`: complete Java classpath for assigned Clusterer; `options`: options to set for a clusterer.
- `clusterDataSet(dataset:String) : String[]`
  Divide data into clusters; `dataset`: instances data; returns: All clusters descriptions.
- `clusterURLDataSet(dataURL:String, dataUser:String, clusteredDataURL:String, clusteredDataUser:String) : String[]`
  Divide data stored in WebDav repository into clusters. Store results also in repository; `dataURL`: an URL of instances; `dataUser`: 'user:password' for reading data; `clusteredDataURL:` address to store clustered data; `clusteredDataUser:` 'user:password' for storing data; returns: An array with clustered data URLs.
- `buildClusterer(dataset:String) : int`
  Cluster dataset and build clusterer; `dataset`: instances data; returns: number of clusters created.
- `clusterData(data:double[][]) : int[]`
  Divides data onto clusters; `data`: data to cluster; returns: an array of clusters' numbers to which every item has been assigned.

WekaFilter
Gem class: `cyfronet.gridspace.gem.weka.WekaFilter`
Description: Methods for filtering input data sets in ARFF format.
Technology: Mocca component, *Weka* library integration, written in *Java*.
Methods:

- `assignFilter(filterPath:String, options:String[])`
  Create a filter object and set options to it; `filterPath`: complete Java classpath for assigned Filter; `options`: options to set for a filter.

- `filterDataSet(dataset:String) : String`
  Filter data; `dataset`: instances data; returns: Instances after filtering.
- `filterURLDataSet(dataURL:String, dataUser:String, filteredDataURL:String, filteredDataUser:String) : String[]`
  Filter data stored in WebDav repository. Store results also in repository; `dataURL`: an URL of instances; `dataUser`: 'user:password' for reading data; `filteredDataURL:` address to store filtered data; `filteredDataUser:` 'user:password' for storing data; returns: URL of filtered data.

WekaURLGem
Gem class: `cyfronet.gridspace.gem.weka.WekaURLGem`
Description: Methods for managing data used in Weka gems, and also for testing classifier prediction.
Technology: Web service, implementation is written in *Java*.
Methods:

- `compare(originalData:String, predictedData:String, attributeName:String)`
  Compare two datasets on selected attribute; returns: prediction quality
- `loadDataFromDatabase(dbUrl:String, query:String, username:String, password:String) : String`
  Load data from database; returns: data file in ARFF format
- `splitData(data:String, trainingDataPercent:Integer) : SplitData`
  Split dataset into training and testing parts; returns: a `SplitData` object with training and testing data as its fields
- `loadDataFromDatabasetoURL(dataUrl:String, dataUser:String, dbUrl:String, query:String, username:String, password:String) : String`
  Load data from database and store it in WebDav repository; returns: stored data URL
- `loadDataFromURL(dataUrl:String, dataUser:String) : String`
  Load data from WebDav repository; returns: data file in ARFF format
- `saveDataToURL(dataset:String, dataUrl:String, dataUser:String) : String`
  Store data in WebDav repository; returns: URL address where data were stored
- `compareURLData(originalURLData:String, originalDataUser:String, predictedURLData:String, predictedDataUser:String, attributeName:String)`
  Compare two datasets that are stored in WebDav repository on selected attribute; returns: prediction quality
- `splitURLData(dataURL:String, dataUser:String, splitDataURL:String, splitDataUser:String, trainingDataPercent:Integer) : SplitURLData`
  Split data stored in repository into training and testing set; returns: a `SplitURLData` object with addresses of training and testing data as its fields

ClutoLib
Gem class: `cyfronet.gridspace.gem.bioinfo.clustering.ClutoLib`
Description: Perform clustering with Cluto software.
Technology: Web service, binary program wrapper, written in *Ruby*.
Methods:

- `cluster(data:Double[], options:Hash) : Integer[]`
  Cluster data, clusterization algorithm should be set in options; returns: a number of cluster to which every data instance has been assigned

ClusterLib
Gem class: `cyfronet.gridspace.gem.bioinfo.clustering.ClusterLib`
Description: Perform clustering with Cluster software.
Technology: Web service, binary program wrapper, written in *Ruby*.
Methods:

- `cluster(data:Double[], options:Hash) : Integer[]`
  Cluster data, clusterization algorithm should be set in options; returns: a number of cluster to which every data instance has been assigned

WekaClustering
Gem class: `cyfronet.gridspace.gem.bioinfo.clustering.WekaClustering`
Description: Perform clustering in Weka library software.
Technology: Local gem, written in *JRuby*, uses *WekaClusterer* gem.
Methods:

- `clusterCobweb(dataset:String, cutoff:Double) : Integer[]`
  Cluster data with Cobweb algorithm; `dataset:` instances data (ARFF format); `cutoff:` cutoff value; returns: a number of cluster to which every data instance has been assigned
- `clusterKmeans(dataset:String, k:Integer) : Integer[]`
  Cluster data with simple K-means algorithm; `dataset:` instances data (ARFF format); `k:` number of clusters to create; returns: a number of cluster to which every data instance has been assigned

PCA
Gem class: `cyfronet.gridspace.gem.bioinfo.dimensions.PCA`
Description: Perform PCA operation on data.
Technology: Local gem, written in *JRuby*, uses *R_gem*.
Methods:

- `pca(data:Double[][], dimension:Integer) : Double[][]`
  perform PCA on data; `data:` input data; `dimension:` a number of dimensions to which input data should be reduced; returns: an array with processed data

MDS
Gem class: `cyfronet.gridspace.gem.bioinfo.dimensions.MDS`
Description: Perform MDS operation on data.
Technology: Local gem, written in *JRuby*, uses *R_gem*.
Methods:

- `mds(data:Double[][], dimension:Integer) : Double[][]`
  perform MDS on data; `data:` input data; `dimension:` a number of dimensions to which input data should be reduced; returns: an array with processed data

C.3. Protein sequence and structure analysis gems

ClustalW
Gem class: `uk.ac.ebi.ClustalW`
Description: Sequence alignment in ClustalW[1].
Technology: External Web service.
Methods:

- `runClustalW(params:Hash, content:Hash) : String`
  Submits a ClustalW job to the service; `params:` an instance of the *inputParams*[2] data structure; `content:` a list of data structures describing the query sequence data; returns: a string containing the job ID (jobid).
- `checkStatus(jobid:String) : String`
  Get the status of a job; `jobid:` the job identifier of the job to check status of; returns: a string indicating the status of the job. Current values are: DONE: job has finished, and the results can then be retrieved; ERROR: the job failed or no results where found; NOT_FOUND: the

[1] The complete API is available at http://www.ebi.ac.uk/Tools/webservices/services/clustalw
[2] http://www.ebi.ac.uk/Tools/webservices/services/clustalw#inputparams

job id is no longer available (job results are deleted after 24 h); PENDING: the job is in a queue waiting processing; RUNNING: the job is currently being processed.

- `poll(jobid:String, type:String) : Base64`
  Wait until the job has finished and get the specified type of result data; `jobid:` the job identifier of the job to check status of; `type:` a string specifying the type of result to retrieve; returns: a base64 encoded string containing the result data.

ClustalW2
Gem class: `uk.ac.ebi.ClustalW2`
Description: Sequence alignment in ClustalW2[3].
Technology: External Web service.
Methods:

- `runClustalW2(params:Hash, content:Hash) : String`
  Submits a ClustalW2 job to the service; `params:` an instance of the *inputParams* data structure; `content:` a list of data structures describing the query sequence data; returns: a string containing the job ID (jobid).
- `checkStatus(jobid:String) : String`
  Get the status of a job; `jobid:` the job identifier of the job to check status of; returns: a string indicating the status of the job.
- `poll(jobid:String, type:String) : Base64`
  Wait until the job has finished and get the specified type of result data; `jobid:` the job identifier of the job to check status of; `type:` a string specifying the type of result to retrieve; returns: a base64 encoded string containing the result data.

Muscle
Gem class: `uk.ac.ebi.Muscle`
Description: Sequence alignment in Muscle[4].
Technology: External Web service.
Methods:

- `runMuscle(params:Hash, content:Hash) : String`
  Submits a Muscle job to the service; `params:` an instance of the *inputParams* data structure; `content:` a list of data structures describing the query sequence data; returns: a string containing the job ID (jobid).
- `checkStatus(jobid:String) : String`
  Get the status of a job; `jobid:` the job identifier of the job to check status of; returns: a string indicating the status of the job.
- `poll(jobid:String, type:String) : Base64`
  Wait until the job has finished and get the specified type of result data; `jobid:` the job identifier of the job to check status of; `type:` a string specifying the type of result to retrieve; returns: a base64 encoded string containing the result data.

TCoffee
Gem class: `uk.ac.ebi.TCoffeeW`
Description: Sequence alignment in T-Coffee[5].
Technology: External Web service.
Methods:

- `runTCoffee(params:Hash, content:Hash) : String`
  Submits a T-Coffee job to the service; `params:` an instance of the *inputParams* data structure; `content:` a list of data structures describing the query sequence data; returns: a string containing the job ID (jobid).

[3] The complete API is available at http://www.ebi.ac.uk/Tools/webservices/services/clustalw2
[4] The complete API is available at http://www.ebi.ac.uk/Tools/webservices/services/muscle
[5] The complete API is available at http://www.ebi.ac.uk/Tools/webservices/services/tcoffee

- `checkStatus(jobid:String) : String`
  Get the status of a job; `jobid:` the job identifier of the job to check status of; returns: a string indicating the status of the job.
- `poll(jobid:String, type:String) : Base64`
  Wait until the job has finished and get the specified type of result data; `jobid:` the job identifier of the job to check status of; `type:` a string specifying the type of result to retrieve; returns: a base64 encoded string containing the result data.

Mammoth
Gem class: `cyfronet.gridspace.gem.structure_comp.Mammoth`
Description: Align sequences based on structure alignment in Mammoth software.
Technology: Web service, binary program wrapper written in *Ruby*.
Methods:

- `compare_structures(pdb_structures:Hash, pdb_ids:Hash, options:String[]) : Hash`
  Run Mammoth for comparing pdb_structures; returns: a map with results, where alignment result type is converter to a map with pdbid as a key and aligned sequence as values
- `get_available_results_types() : Hash`
  returns: names and descriptions of available result types to download

MultiProt
Gem class: `cyfronet.gridspace.gem.structure_comp.MultiProt`
Description: Align sequences based on structure alignment in MultiProt software.
Technology: Web service, binary program wrapper written in *Ruby*.
Methods:

- `compare_structures(pdb_structures:Hash, pdb_ids:Hash, options:String[]) : Hash`
  Run MultiProt and Staccato for comparing pdb_structures and creating sequences alignment; returns: a map with results, where alignment result type is converter to a map with pdbid as a key and aligned sequence as values

SSM
Gem class: `cyfronet.gridspace.gem.structure_comp.`Ssm
Description: Multiple structures alignment in Secondary Structure Matching service[6].
Technology: Web service, HTTP communication wrapper written in *Ruby*.
Methods:

- `compare_structures(pdb_structures:Hash) : Hash`
  Creates request to SSM server and handles complete communication; returns: an sequence alignment based on structures alignment in FASTA format

Dali
Gem class: `uk.ac.ebi.Dali`
Description: Pairwise structure alignment in DaliLite software[7].
Technology: External Web service.
Methods:

- `runDaliLite(params:Hash) : String`
  Submits a DaliLite job to the service; `params:` an instance of the *inputParams* data structure with structures to alignment; returns: a string containing the job ID (jobid).
- `checkStatus(jobid:String) : String`
  Get the status of a job; `jobid:` the job identifier of the job to check status of; returns: a string indicating the status of the job.
- `poll(jobid:String, type:String) : Base64`
  Wait until the job has finished and get the specified type of result data; `jobid:` the job

[6] http://www.ebi.ac.uk/msd-srv/ssm/ssmstart.html
[7] The complete API is available at http://www.ebi.ac.uk/Tools/webservices/services/dalilite

identifier of the job to check status of; `type:` a string specifying the type of result to retrieve; returns: a base64 encoded string containing the result data.

ClustalWUtils
Gem class: `cyfronet.gridspace.gem.structure_comp.ClustalWUtils`
Description: Computing W score and W profile for aligned sequences.
Technology: Web service, service implementation is written in *Ruby*.
Methods:

- `get_complete_sequences(aligned_sequences:String) : Hash`
  Analyze aligned sequences and concatenate appropriate strings into one for every found identifier, it is useful for parsing results from various services; returns: a map with sequence identifier as key and complete aligned sequence as value
- `get_w_profiles_for_proteins(w_score:Double[], aligned_sequences:String[]) : Double[][]`
  Count W profile for every aligned sequene; returns: an array of W profile values, in the same order as aligned_sequences were provided.
- `get_w_score_for_sequences(aligned_sequences:String[], average_window:Integer, codes:Integer) : Hash`
  Count W score for aligned sequences; `average_window:` a length of averaging window; `codes:` number of codes that are valid for aligned sequences; returns W score values.

C.4. Protein binding site prediction gems

CastP
Gem class: `cyfronet.gridspace.gem.pocket.CastP`
Description: CASTp [17, 27] is an online tool that locates and measures pockets and voids on 3D protein structures. It is based on the alpha shape and the pocket algorithm developed in computational geometry. In CASTp, voids are defined as buried unfilled empty space inside proteins after removing all hetero atoms that are inaccessible to water molecules (modeled as a spherical probe of defined radius, 1.4 A) from outside. Pockets are defined as concave surface with constrictions at the opening on the surface regions of proteins. Unlike voids, pockets allow easy access of water probes from the outside.
Technology: Web service, HTTP communication wrapper, implementation is written in *Ruby* with Task queuing system mechanism.
Methods:

- `get_results_from_file(filename:String, filebody:String, options:Hash) : String`
  Submits a pdb file request to the service; `filename:` file identifier; `filebody:` content of the pdb file; returns: a string containing the task ID (taskid).
- `get_results_from_pdbid(pdbid:String, options:Hash) : String`
  Submits a pdb id request to the service; `pdbid:` structure identifier; returns: a string containing the task ID (taskid).
- `get_status(taskid:String) : Integer`
  Get the status of a task; `taskid:` the task identifier (obtained from one of the computation request methods); returns: a Integer indicating the status of the job (0 – *CREATED*; 1 – *QUEUED*; 2 – *PROCESSING*; 3 – *FINISHED*).
- `get_results(taskid:String) : ProteinTaskResult`
  Download results from service if task is in *FINISHED* status; returns: TaskResult object which may be *OK* or *ERROR* type.
- `get_available_options() : Hash`
  List available options; returns: A map with option name as a key and option description as a value.

ConSurf
Gem class: `cyfronet.gridspace.gem.pocket.ConSurf`
Description: ConSurf [43] calculates conservation scores for residues of a given 3D - structure of a protein or a domain. It is designed to identify hot spots and surface patches that are likely to be in contact with other proteins, domains, peptides, DNA, RNA or ligands. The underlying assumption is that key residues that are important for binding should be conserved throughout evolution, just like residues that are crucial for maintaining the protein fold. First, the sequence is extracted from the Protein Data Bank (PDB) file and used as a template in search for close homologues using PSI-BLAST. Selected hits (using Expectation value and Maximum Number of Homologues as criterion) are being aligned using Multiple Sequence Alignment (MSA) program. The MSA is used to build a phylogenetic tree using the neighbor joining (NJ) algorithm. The conservation scores are calculated using either an empirical Bayesian or the Maximum Likelihood method. There is also ConSurf Database (ConSurfDB) that offers precalulted results for entries from PDB.
Technology: Web service, HTTP communication wrapper, implementation is written in *Ruby* with Task queuing system mechanism.
Methods:

- The same as *CastP*

Fod
Gem class: `cyfronet.gridspace.gem.pocket.Fod`
Description: FOD [22] calculates distribution of hydrophobicity in terms of deficiency/excess interpreted as a measure of structural and functional specificity. The observed distribution of hydrophobicity in the protein body is compared to the idealized one expressed by a three-dimensional Gauss function. The differences between these two quantities calculated for each residue in a structure are presented as $\Delta\tilde{H}$ profile. Residues with high $\Delta\tilde{H}$ values are expected to form ligand binding site(s).
Technology: Web service, Binary program wrapper, implementation is written in *Ruby*.
Methods:

- The same as *CastP,* except `get_results_from_pdbid` - no available

Ligsite_csc
Gem class: `cyfronet.gridspace.gem.pocket.Ligsite_csc`
Description: LIGSITEcsc [37] is an extension and implementation of the LIGSITE algorithm. LIGSITEcsc is based on the notion of surface-solvent-surface events and the degree of conservation of the involved surface residues. First, instead of capturing protein-solvent-protein events, as it is in the case of LIGSITE method, surface-solvent-surface events are being captured using the protein's Connolly surface. Second, the pockets identified are re-ranked by the degree of conservation of the involved surface residues.
Technology: Web service, HTTP communication wrapper, implementation is written in *Ruby* with Task queuing system mechanism.
Methods:

- The same as *CastP*

Pass
Gem class: `cyfronet.gridspace.gem.pocket.Pass`
Description: PASS [18] is a simple computational tool that uses geometry to characterize regions of buried volume in proteins and to identify positions likely to represent binding sites. The PASS algorithm is designed to fill the cavities in a protein structure with a set of spheres and to identify a few of these spheres (Active Site Points, ASPs) that most likely represent the centers of binding pockets. Crevice filling is performed in layers using three-point Connolly-like sphere geometry. An initial coating of probe spheres is calculated with the protein as substrate, then additional layers of probes are accreted onto the previously found probe spheres. Only probes with low solvent

exposure are retained, and the routine finishes when an accretion layer produces no new buried probe spheres.
Technology: Web service, Binary program wrapper, implementation is written in *Ruby*.
Methods:

- The same as *CastP,* except `get_results_from_pdbid` - no available

PocketFinder
Gem class: `cyfronet.gridspace.gem.pocket.PocketFinder`
Description: Pocket-Finder implements LIGSITE [34] which is based on POCKET algorithm but circumvents most of its drawbacks. LIGSITE is a method for the automatic detection of pockets on the surface of proteins that may act as binding sites for small molecule ligands. In the first stage a regular Cartesian grid is generated. Next, grid points are labeled as solvent or protein depending whether they are accessible or inaccessible to solvent modeled by a probe sphere (radius set to 1.4 Å). Then for each point 7 directions (x, y, z and four cubic diagonals) are scanned in order to identify protein-solvent-protein (PSP) events. Pockets and cavities are defined as regions of grid points with a defined minimum number of PSP events.
Methods:

- The same as *CastP*

QSiteFinder
Gem class: `cyfronet.gridspace.gem.pocket.QSiteFinder`
Description: QSite-Finder [45] is a method for ligand binding site prediction. It uses the interaction energy between the protein and a simple van der Waals probe to locate energetically favourable binding sites. Energetically favourable probe sites are clustered according to their spatial proximity and clusters are then ranked according to the sum of interaction energies for sites within each cluster.
Methods:

- The same as *CastP*

SuMo
Gem class: `cyfronet.gridspace.gem.pocket.SuMo`
Description: SuMo [40] is a method designed to detect similar three-dimensional (3D) sites in proteins. First, the PDB file containing the atomic coordinates for a protein structure is converted into a data structure suiTable for fast comparison. Then the comparison step follows, using pre-formatted data that may come from this database. The basis for this method is a representation of the protein structure by a set of stereochemical groups that are defined independently from the notion of amino acid. Heuristics for finding similarities uses graphs of triangles of chemical groups to represent the protein structures.
Methods:

- The same as *CastP,* except `get_results_from_pdbid` - no available

WebFeature
Gem class: `cyfronet.gridspace.gem.pocket.WebFeature`
Description: WebFEATURE [46] is a structural analysis tool that allows to scan query structures for functional sites in both proteins and nucleic acids. It is the public interface to the scanning algorithm of the FEATURE package, a supervised learning algorithm for creating and identifying 3D, physicochemical motifs in molecular structures. Given an input structure or Protein Data Bank identifier (PDB ID), and a statistical model of a functional site, WebFEATURE returns rankscored hits in 3D space that identify regions in the structure where similar distributions of physicochemical properties occur relative to the site model.
Methods:

- The same as *CastP*

ResultsConverter
Gem class: `cyfronet.gridspace.gem.pocket.ResultsConverter`
Description: Conversion predicted binding site description from a service specific format to the common format.
Technology: Web service, service implementation is written in *Ruby*.
Methods:

- `get_atoms_for_pocket(service_name:String, results:String, name:String, options:Hash) : String`
  Get residue numbers for predicted binding site. Suitable only for one–chain protein.
- `get_atoms_and_chains(service_name:String, results:String, name:String, options:Hash) : String`
  Get residue numbers and chain identifiers for predicted binding site.
- `get_atoms_for_chain(service_name:String, results:String, name:String, options:Hash, chain:String) : String`
  Get residue numbers and chain identifiers for the most probably binding site that contains at least one residue that belongs to the selected `chain`.
- `get_available_converters() : String[]`
  List available format converters.

C.5. Results presentation gems

Gnuplot
Gem class: `cyfronet.gridspace.gem.GnuPlot`
Description: Create plots from numerical data.
Technology: Web service, binary program wrapper in *Ruby*.
Methods:

- `plot(params:String, inData:String) : String`
  Create plot with one input data file; `params:` GnuPlot script with '*input*' string means the place to put path to data file; `inData:` content of input data file; returns: an URL to the created plot.
- `multiplot(params:String, inputs:String[]) : String`
  Create plot with mutliple input data files; `params:` GnuPlot script with '*inputNR*' string means the places to put path to data files (*NR* can not be greater than size of *inputs* array); `inputs:` an array with content of input data files; returns: an URL to the created plot.

ProteinWorkshop
Gem class: `cyfronet.gridspace.gem.bioinfo.visualize.ProteinWorkshop`
Description: Create visualization of protein structure in ProteinWorkshop software.
Technology: Web service, Java Web Start generator, written in *Ruby*.
Methods:

- `visualize_structure(structure_path:String, options:Hash) : String`
  Creates JNLP file for ProteinWorkshop application. The path to the visualized protein, `structure_path` , is set as program argument.

Jmol
Gem class: `cyfronet.gridspace.gem.bioinfo.visualize.Jmol`
Description: Create visualization of protein structure in Jmol software or visualize many structures at once in enhanced Jmol application.
Technology: Web service, Java Web Start generator, written in *Ruby*.
Methods:

- `visualize_structure(jmol_script:String, options:Hash) : String`
  Create JNLP file to Jmol application. A path to the jmol_script is set as an application argument.
- `visualize_structures(xml_data:String, options:Hash) : String`
  Create JNLP file to the developed wrapper for Jmol visualization. A path to XML script that contains information about predicted binding sites is set as an application argument.
- `create_jmol_script(results:Hash) : String`
  Create Jmol script from predicted binding sites. Input is a map with service name as a key and results from this service as a value.
- `create_xml_script(results:Hash) : String`
  Create XML script from predicted binding sites. Input is a multi–stage map file_id => service => predicted site

JalView
Gem class: `cyfronet.gridspace.gem.bioinfo.visualize.JalView`
Description: Create visualization of sequences alignment in JalView.
Technology: Web service, Java Web Start generator, written in *Ruby*.
Methods:

- `visualize_sequences(alignment_path:String, options:Hash) : String`
  Creates JNLP file for JalView application.

JTreeView
Gem class: `cyfronet.gridspace.gem.bioinfo.visualize.JTreeView`
Description: Create visualization of microarray data in JTreeView.
Technology: Web service, Java Web Start generator, written in *Ruby*.
Methods:

- `visualize_data(files_paths:String[], options:Hash) : String`
  Creates JNLP file for JTreeView application. Paths to *cdt*, *gtr* and *atr* files are required.