# *Integration and Visualisation of Data in Bioinformatics*

a dissertation presented
by
Gustavo Adolfo Salazar Orejuela
to
The Computational Biology Group
Supervised By
Dr. Nicola Mulder

in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy
in the subject of
Bioinformatics

University of Cape Town
Cape Town, Western Cape, South Africa
December 2015

Thesis Supervisor: Dr. Nicola Mulder Gustavo Adolfo Salazar Orejuela

# *Integration and Visualisation of Data in Bioinformatics*

*Abstract*

One day get here!

# Contents

# Listing of figures

*Pá mi nena*

# Acknowledgments

*What a hike!*

I'm ready for this, it is going to be a short and pleasant walk, with nice views and beautiful landscapes, what can go wrong? if with the hike I just finished I pretty much have come half way, the rest should be easy, no surprises!

It has been a long time and Im still on my way, but I'm not complaining, why would've I? Besides, I wasn't that wrong. Yes, it wasn't short, and yes there were (are) scary moments, and yes, not everything was pretty, but you have to go through repetitive dusty hills if you want to get to the viewpoint; and at the end, it is only possible to appreciate the view, when you recognise the effort that you have put into it, that feeling of achievement is mine, no one can't take it away.

But clearly I wouldn't be here if I was by myself. The path wasn't always clear and there were times that I felt lost, how many times I have to stop for directions? way to many! And no one like Nicky to calmly show me that I wasn't lost, that there was a path in front of me, that maybe I should just have some water, breath and keep walking. Thanks for your help and support Nicky.

A lesson learnt is the importance of a good base camp, a warm place where you can rest after an extraneous day, and if you are lucky enough you might find friends in those sort of places, the kind of friends who give you a ladder to brake into your own house, or talk to you even when they barely understand you. Thanks Kate, Louis, Becca, Michelle and Richard.

Sometimes I ask myself, how I end up committing to this challenge, and then realise that the question is not how, but who motivate me to do it, and here the list is long, but in the top of it is my family, because they teach me to walk, sure they also gave me the boots and the raincoat, but that is worth nothing if i didn't know how to walk. My mom who worries about a thousand details that I didn't consider when I gave the first steps, and my dad who would give everything away, to get a chopper if I need to be rescued. How can I consider myself brave about coming here, if I'm a phone call away of a rescue chopper? And

my brother, oh Juancho, how I wish you were here, running over the same old ground. Gracias Má! Gracias Pá! Gracias Juancho!

I knew this was a technical hike, but I'm lucky enough to know some people who has walk similar ones, and their invaluable advise were always there. Starting from Rafa, who gave me a place to stay in the first camp I found, showed me around, and allowed me to safely explore during those cold times, he also introduced me to Henning, who for a short period guided me and allowed me to be part of his group, in which I have the pleasure of met so many professional hikers, from whom I hope I have learned one or two tricks.

But you can't walk all the time, and there were quite a few friends that i met in that cold camp, with who I played, sang and danced. Very good memories from Ruben, Claudia, Jose, Bernat, Leyla, Fernanda, Ana, Josue and Jorge.

Back in the base camp I met who have become my partner in crime, the person that walked by my side ever since, the one who would take care of me if I twist an ankle, even if I have make her upset. She have read this whole story and have give me lots of advices (sic) to make it better, so, if this whole thing is readable is only because of her. Thanks Julie, thanks mi nena, I really hope this is only the first of many adventures together.

There were hills bigger than my skills, but then I was lucky enough to see that by collaborating the obstacles are easy to deal with, Thanks again to Rafa, but also to Manuel, Seb, Alex, Leyla, Jose, Andy, Jon and all the people of the BioJS and DAS groups. But the group that give me the most support during this time was CBIO, in which I always found people willing to help me, to whom I shared experiences with, making my trip easier, and hopefully me making theirs better. We might not all been walking to the same place, but we definitely share quite a few paths. Thanks Ayton, Holy, Gaston, Kenneth, Jon, Renaud, Rebone and everyone in the group.

So many memories build during this campaign, for example to find a couple of waterfalls in side paths, or to be able to stop and have a pineapple in the road with Nats and Jon, and sharing it knowing that our friends really appreciate a good pineapple, wouldn't you Danie, Brett, Tamsin, Sean, Sarah, Nats, Ayton and Julie?? You guys rock by the way.

And now here at top of this hill, I can only feel grateful, It is like scoring a goal with the defenders on you and the goalie centimetres away of the ball, and you are calm enough to hit the ball, gently but firmly, strong enough to elevate it over the body of the keeper, but soft enough to make it fall in the middle of the three posts. It is yours, yeah! but who made the pass?, and who stole the ball in the first place? and who were the warriors defending the whole time?, and who made the fancy moves just for the show? Simply, your friends, your team. Goal!

# Abbreviations

**2DE** . . . . . Two-dimensional gel electrophoresis
**API** . . . . . Application Program Interface
**BAM** . . . . . Binary Alignment/Map
**BBI** . . . . . Big Binary Indexed
**BigWIG** . . . . . Big Binary Indexed Wiggle format
**BioPAX** . . . . . Biological PAthway eXchange
**BRENDA** . . . . . BRaunschweig ENzyme DAtabase
**DAS** . . . . . Distributed Annotation System
**DIGE** . . . . . Differential Imaging Gel Electrophoresis
**DNA** . . . . . Deoxyribonucleic acid
**EBI** . . . . . European Bioinformatics Institute
**EMBL** . . . . . European Molecular Biology Laboratory
**GO** . . . . . Gene Ontology
**HPRD** . . . . . Human Protein Reference Database
**HTTP** . . . . . Hyper Text Transfer Protocol
**HTML** . . . . . Hyper Text Markup Language
**HUPO** . . . . . HUman Proteome Organisation
**ICA** . . . . . Independent Component Analysis
**IGV** . . . . . Integrative Genomics Viewer
**KEGG** . . . . . Kyoto Encyclopedia of Genes and Genomes
**LIMS** . . . . . Laboratory Information Management System
**MDS** . . . . . Metrical multi-Dimensional Scaling
**MIAPE** . . . . . Minimum Information About a Proteomics Experiment
**MS** . . . . . Mass Spectrometry
**NAR** . . . . . Nucleic Acids Research
**NCBI** . . . . . National Center for Biotechnology Information
**NGS** . . . . . Next Generation Sequencing

**NMDS** . . . . . Non-Metrical multi-Dimensional Scaling
**OMIM** . . . . . Online Mendelian Inheritance in Man
**OWL** . . . . . Web Ontology Language
**pb** . . . . . pairbase
**PCA** . . . . . Principal Components Analysis
**PLS** . . . . . Partial Least Squares
**PRIDE** . . . . . PRoteomics IDEntifications database
**PSI** . . . . . Proteomics Standards Initiative
**PSICQUIC** . . . . . PSI common query interface
**PTM** . . . . . Post Transactional Modifications
**QS** . . . . . Quality Scores
**RDBMS** . . . . . Relational Database Management Systems
**RDF** . . . . . Resource Description Framework
**REST** . . . . . REpresentational State Transfer
**RIA** . . . . . Rich Internet Applications
**RNA** . . . . . Ribonucleic acid
**ROI** . . . . . Region Of Interest
**SAM** . . . . . Sequence Alignment/Map
**SADI** . . . . . Semantic Automated Discovery and Integration
**SCOP** . . . . . Structural Classification of Proteins
**SOAP** . . . . . Simple Object Access Protocol
**SOM** . . . . . Self-Organised Maps
**SNP** . . . . . Single Nucleotide Polymorphism
**SPARQL** . . . . . Query Language for RDF
**SRS** . . . . . Sequence Retrieval System
**SQL** . . . . . Standard Query Language
**UCSC** . . . . . University of California, Santa Cruz
**URL** . . . . . Universal Resource Locator
**WIG** . . . . . Wiggle format
**W3C** . . . . . World Wide Web Consortium
**WSDL** . . . . . Web Service Description Language
**XML** . . . . . Extensible Markup Language

*If I have seen further it is by standing on the shoulders of giants*

Isaac Newton

# 1 Introduction

*Bioinformatics by it's very nature is multi–disciplinary*: biotechnology methods are used to capture data, mathematical and statistical models are necessary to make sense of it, and software/hardware is necessary to process all the information, all of which should be seen from the viewpoint of existing biological knowledge.

Techniques and methods of uncovering the logic behind cellular processes are in constant development. The success of these methods is usually related to the ability to isolate a particular state/measurement of the organic material (e.g. level of expression of a single protein, number of mutations of a gene in a population, etc.). However, cellular functions of a higher level and organism's processes can only be studied with a more holistic approach.

Integrating information from different experiments in the context of the current knowledge-base is one of the most recurrent duties that researchers have to accomplish. This task however takes on another dimension when the goal is to connect related data sets in a general way. For example, a researcher usually needs to connect the target gene of one experiment with the information about the protein in which it is known to be expressed. However this becomes far more difficult if the aim is to automatically connect a whole dataset of genes and their protein products.

The challenges of integrating heterogeneous datasets range from technical (e.g. incompatibilities of

the storage systems) to structural (e.g. two datasets can refer to the same entity using different identifiers), but overall the major challenge is to not lose the meaning of the connection e.g. linking proteins and genes, and knowing what their relationship is.

Integration of data usually refers to connecting data at a low level, by means of storing aggregated information from several sources or by saving links to where the source is. However it is also possible to integrate it at a higher level, where information is not saved and the aggregates are built and visualized on demand.

This project explores both the integration and visualization of information in bioinformatics; the rest of this introduction presents both the state of the art and fundamental technologies in both fields. Chapter 2 presents the efforts made during this doctorate that contribute to the methods on how data are integrated in bioinformatics projects. Chapter 3 describes our inputs to the visualization of data in bioinformatics, focusing in particular on a web tool for the visualization of protein-protein interactions. The final chapter contains the conclusions of the project.

## 1.1 Integration of information

### 1.1.1 State of the art

The latest version of the Nucleic Acids Research NAR Database Issue added a further 58 databases to the online collection held by the journal, which then reached the number of 1552 databases [25]. This collection is far from including every single database, but it is a good reflection of the number of available resources. The approaches to integrate data from all these sources are themselves heterogenous, and are focused on different types of integration, from simply linking resources to development of complex structures of aggregated information. In [36] the authors categorise the different techniques used to integrate data in bioinformatics into eight approaches, and then these categories were reorganised in [125] into: data warehousing, federated databasing, service oriented integration, semantic integration and Wiki-based integration.

We now present some of the most representative projects that have attempted to integrate data and offer a solution for this requirement in the bioinformatics field, by using one or more of the aforementioned approaches. [1]

[1] We have decided to include the description of the Distributed Annotation System in a separate section of this document, because it is the base-technology of the developments shown in Chapter 2, and therefore it requires an extended description.

*Data warehousing*

It is centralised repository where the information from different sources is copied and processed to be kept in a single place providing a single access point to their data. However the preprocessing of the data is usually a complex process and the posterior additions or editions might require a lot of work.

BioWarehouse

BioWarehouse is an open source toolkit to create data warehouses using MySQL or Oracle [63]. The motivation behind this project is to provide a single access point that supports Standard Query Language SQL running in a high performance environment. This projects follows the data warehousing approach to integrate data, however the authors argue that it can be used as a part of a federated system and therefore, it doesn't aim to replace existing distributed systems but rather to complement them.

The development efforts were focused on the creation of a relational data model that supports the information from several biological entities. Figure 1.1.1 shows the main datatypes that are defined in the BioWarehouse scheme including: Taxon, BioSource, Nucleic Acid, Gene, Protein, Feature, Reaction, Chemical and Pathway. It was the objective of the creators that the model evolves to include other entities, but at the same time keeps the model as simple as possible.

The package that comprises BioWarehouse includes a set of loaders implemented in java and C++, that allows the automation of the process of loading data from several popular sources.

An instance of BioWarehouse called Publichouse is available online `https://publichouse.ai.sri.com/phpmyadmin/` (last checked December 2014) and provides access to compiled data from: NCBI Taxonomy, Enzyme, MetaCyc Chemical Compound Ontology, MultiFun Gene Ontology, MetaCyc Pathway Ontology, BioCyc, Swiss-Prot and TrEMBL.

The most recent contribution reported on their website is from August 2010 and includes the installation of the web based interface for mysql databases called myphpadmin. This highlights the lack of development that the project has had in the last 4 years. A similar situation has been observed in projects that follow the same warehousing strategy in bioinformatics. Most of them were published around the same period of time but have been inactive in recent years, or have broken links to the tool, for example, Atlas [102] published in 2005, LCB[4] published in 2006 and M-Chips [24] from 2002. From our research, the only active project on infrastructure of data warehousing for biological data is BioDWH, which is described below.

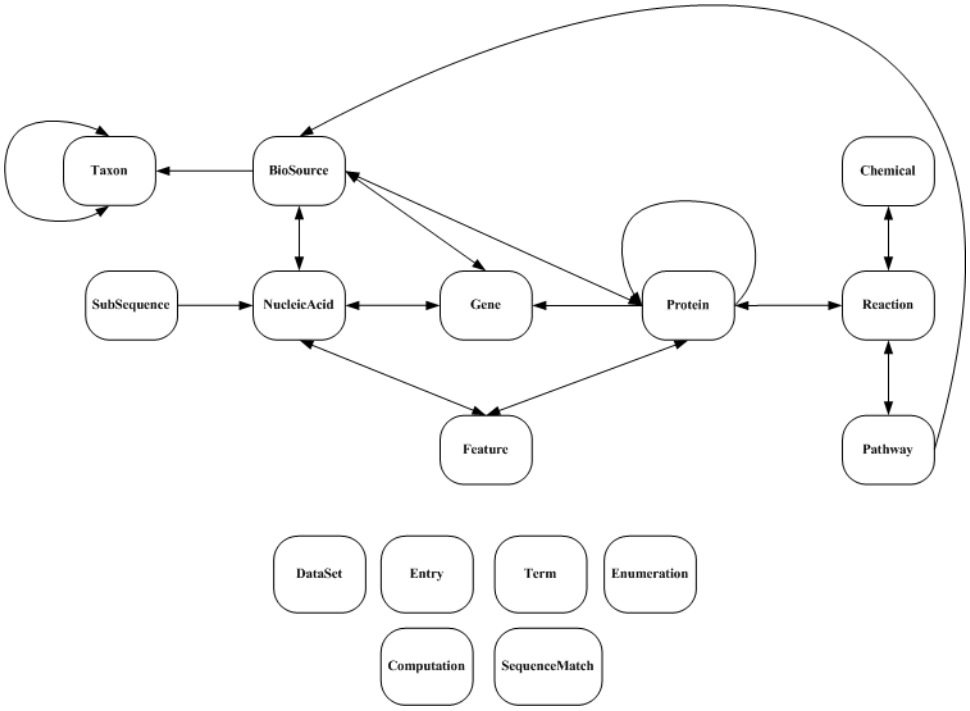


**Figure 1.1.1:** The main datatypes in the BioWarehouse schema, and the relationships between them.

BioDWH

The data warehouse for life science data integration known as BioDWH is a project developed at the Bielefeld University. BioDWH is a Java based project that developed an object-relational mapping using the library Hibernate to connect to the most common relational database management systems RDBMS (e.g. MySQL, Oracle, PostgreSQL) in order to create a centralised repository that integrates information from various biological databases [112].

BioDWH's main objective is to increase customisation of the data warehouse concept improving performance, scalability and having quality data up to date. As part of the project they have included some parsers to extract information from well-known available resources (e.g. UniProt, KEGG, OMIM, etc.). Figure 1.1.2 presents the extensions to a general data warehouse design done in this project in order to define an architecture oriented to life sciences data.

It is important to highlight the inclusion of a monitoring system that keeps track of the need for updates from the different sources. Instead of direct access to the RDBMS, BioDWH provides an API that can be queried remotely, and a Graphical User Interface that enables the configuration of the different components including monitors and parser along with the queuing and displaying of content.

Two projects have been reported in the literature that are using BioDWH: DAWIS-M.D. and

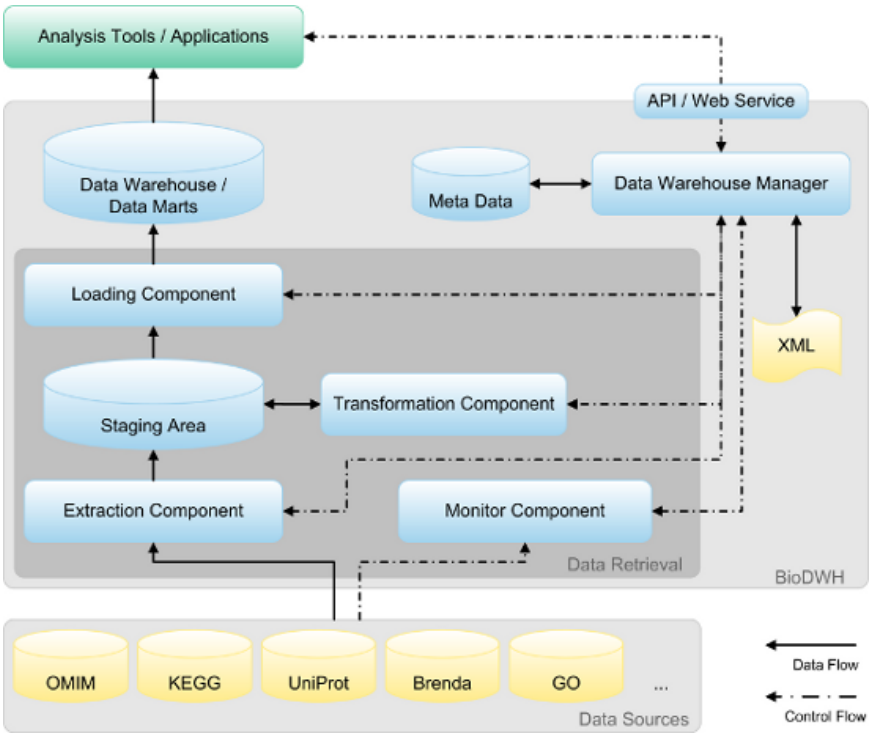


**Figure 1.1.2:** BioDWH System Architecture.

VANESA. DAWIS-M.D. is oriented to metabolic data and integrates eleven different databases: BRENDA, EMBL, HPRD, KEGG, OMIM, SCOP, Transfac, Transpath, ENZYME, GO and UniProt [41]. VANESA uses DAWIS-M.D. in order to access important information for the modelling of biological processes and systems as biological networks [10].

BioMart

BioMart started by using the same principle as BioWarehouse, BioDWH and other projects of data warehousing in life science: "*to create one universal software system for biological data management and empower biologists with the ability to create complex, customised datasets*" [54].

With this object in mind, BioMart has grown from an extension to the Ensembl website for data mining, to become an international effort for the integration of biological data. This has been achieved by first defining a general software infrastructure for further customisation, and then extending this architecture in order to support multi-database repositories as a data federation system, where all the entities use a predefined relational schema that is generic enough to support any kind of data.

Given the inclusion of a multi-database paradigm, the BioMart team developed a server that provides access to a collection of sources called a BioMart Portal. Figure 1.1.3 shows the two possible

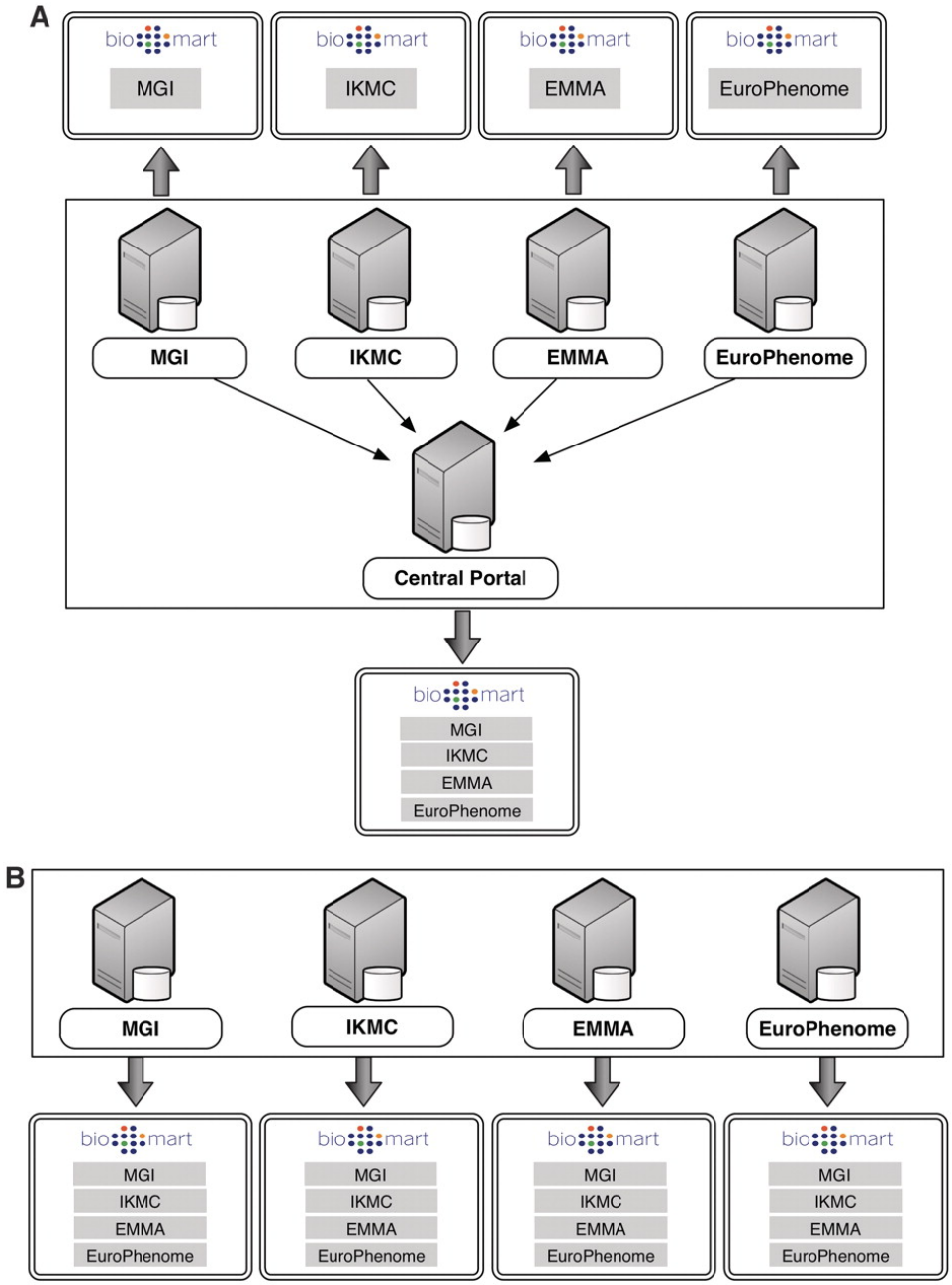


**Figure 1.1.3:** Biomart Portal Architecture. The portal can be set either with a master/slave or with a peer-to-peer configuration.

configurations of the BioMart Portal. In configuration (a) each data source serves their own data and an independent server works as a central portal generating a unified view of the whole system. Alternatively, in configuration (b) all the data sources are treated as peers and they communicate with each other in order to be able to provide not only their own data, but also their peer's data.

BioMart includes a tool to automatically transform any 3rd form normalised database schema into the reverse-star scheme type used by this system. Once the dataset has been transformed, BioMart provides different view ports to it: Web-based Graphic Interface, Restful services and API connectors using Java [54].

*Federated databasing*

This approach requires the agreement of multiple sources to follow a similar structure in order to allow a standard query over several instances. By dealing with smaller datasets than the data warehousing approach, the complexity of post-processing is simplified, however it requires that the providers deal with the extra work of maintaining their data as the federated database agreement establishes.

The HUPO Proteomics Standards Initiative

The Proteomics Standards Initiative (PSI) is a collaborative initiative run by volunteers and coordinated as a work group of the HUman Proteome Organisation (HUPO), whose object is to define standards to enable capture, comparison, exchange and verification of proteomics data. [40]. PSI is the result of a common effort from the interested parties in the proteomics domain.

This effort can be categorised into three major infrastructure elements: (1) A specification of the Minimum Information About a Proteomics Experiment (MIAPE), (2) data exchange formats that are compliant with MIAPE, and (3) the use of controlled vocabularies (CV) in order to ensure consistency on the data content. In this way the proposed standard can be stable while its content can evolve by updating the CV.

These recommendations are not intended to specify the methods and procedures for proteomics experiments, and should be seen as reporting guidelines.

This initiative has been growing for the last 10 years, and now its standards are widely adopted in the proteomics community. The proposed specifications cover different branches of the field, for instance the PSI-MI formats were defined to deal with Molecular interaction data, PSI-MS works on standards for mass spectrometry data and PSI-MOD focuses on protein modifications.

However even if all the parts follow the recommendations, a strategy to integrate this data is required. It is for this reason that the PSI common query interface (PSICQUIC) was created: a community standard that enables programmatic access to molecular-interaction data resources [5]. This proposal

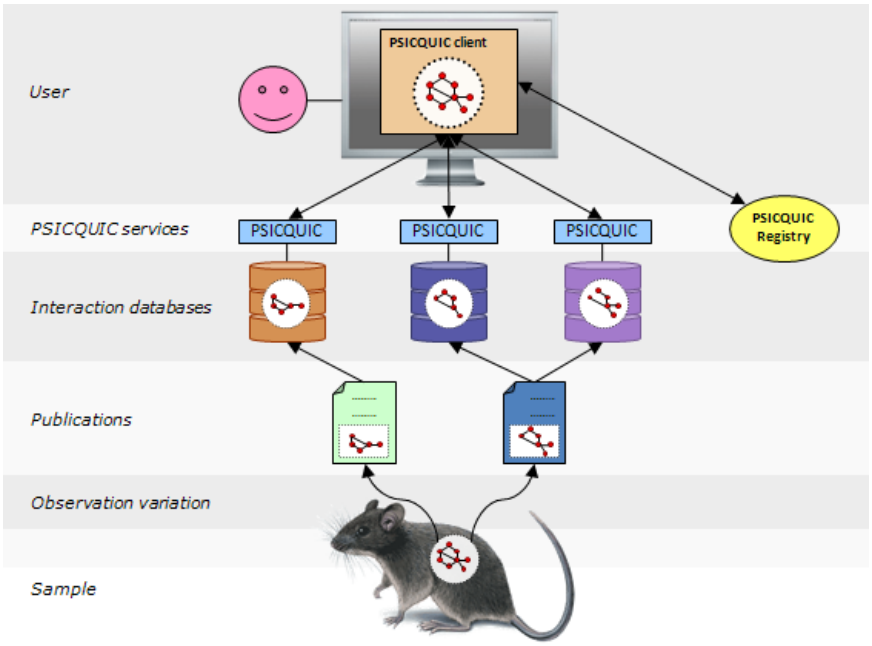


**Figure 1.1.4:** PSICQUIC Architecture.

includes a query language (MIQL) and an architecture to execute a query on distributed sources. Figure 1.1.4 shows the architecture of PSICQUIC. The idea is that several samples from an organism can be processed by different experiments, and their findings can be published in independent articles, which consequently can be stored in more than one interaction database. PSICQUIC proposes that the providers include an extra layer to access this information, which can be queried using MIQL and with responses following the PSI-MI formats. In this way a client can use a single query over multiple resources and create a unified image.

The PSI group recognises the importance of providing software tools to promote the adoption of the proposed standards. The most recent implementation of the PSICQUIC web service was released when version 1.3 of the PSICQUIC specification was made public in 2013 [18]. The providers of molecular interaction data are now asked to implement a number of methods in order to support both SOAP and REST web services.

The web service methods will generally accept a MIQL query as input and generate an output in either PSI-XML or PSI-MITAB in its most recent versions. This implementation is based on the Apache Solr indexing software (`http://lucene.apache.org/solr/`), and is reflected in the constitution of MIQL, which is an extension of the Lucene query language used in Solr. This implementation is freely available for any provider who wishes to share their data to the PSI community.

Besides the server, PSICQUIC also provides client libraries to facilitate access to the information from

three different programming languages: Java, Perl and Python. There are ready to use clients, for example the PSICQUIC View accessible from http://www.ebi.ac.uk/Tools/webservices/psicquic/view/main.xhtml or plugins for Cytoscape and the R Bioconductor package.

The last component of the PSICQUIC architecture is the registry, in which information about the available providers is stored as tags. This metadata can be used to query the registry as a RESTful service in order to facilitate the discovery of molecular-interaction data providers [18].

*Service oriented integration*

The principle of service oriented integration is to have atomic components that offer a basic service. These services can then be connected in order to orchestrate the execution of a bigger task. Services can be data providers but can also be programs that execute specific functions.

The definition of a protocol for requests and responses to obtain data or execute services is the key to facilitate the connectivity between the software components. The two standards most widely used are SOAP (Simple Object Access Protocol) services and REST (REpresentational State Transfer) services, where the latter is gaining momentum because of its simplicity. However service oriented integration also requires a large commitment from providers in creating and maintaining the services, and also increasing and maintaining the specifications for the different domains. A comparison between SOAP and REST web services can be found in [82].

Alternatively, the services can run on a local installation, which does not follow the principle of a unique remote resource executing a service to multiple consumers. Nonetheless, this option has been widely used in bioinformatics environments in order to have more control over the data and services offered.

Sequence Retrieval System

The Sequence Retrieval System, better known in bioinformatics as SRS, was probably the most successful project before the introduction of Next Generation Sequencing (NGS) technologies, originally aimed at facilitating access to biological sequences databases [23]. It grew to become an integration system of both data retrieval and applications for data analysis.

SRS was developed following an object-oriented design with the strategy of taking advantage of raw text files, that were the *de-facto* standard on molecular biology analysis. By dealing only with text files, SRS was getting faster retrieval speeds and saving storage space, mainly because data was neither stored nor parsed, only indexed [123].

The indexes obtained were linked via meta-data, offering the user access to the original source plus links to any conceptually related database. This was then presented in a web-based interface. The

automatically created interface for different sources and applications can be complicated for beginners, however SRS provides ways to create customised interfaces.

Probably the most important instance of SRS was the one installed at the European Bioinformatics Institute (EBI), which was used to provide access to the major databases produced and maintained at the EBI. However by December 2013 the service was decommissioned as the service was considered redundant with the efforts to maintain multiple web services. Currently SRS technology is the property of Instem™ and a list of available servers can be found at http://bioblog.instem.com/download/srs-parser-and-software-downloads/public-srs-installations/.

### Taverna

Taverna was originally conceived of as a tool for non expert programmers to design, execute and share workflows of web services [45]. Bioinformatics web services can be a way of providing the available data in a data source (e.g. the PSI services), but traditionally, web services are seen as a remote software component that receives some input data, processes it and generates output data. The main advantage of using web services is that most of the processing load is delegated to the service provider, allowing small groups to run high throughput analysis in remote but powerful machines.

Taverna is a tool that allows the user to create "recipes" of combined web services or pre-composed workflows to execute a higher level computational experiment. The Taverna workbench is a graphical interface that allows the design of workflow, that can be executed either on the same workbench or in independent runner tools such as the Taverna command-line application, the Taverna server or the Taverna lite installation.

By 2013 the Taverna project was reported to have access to over 8000 service operations [122]. With such a large number of resources, there is a necessity to make them searchable. This is the goal of the BioCatalogue: a registry for web services where both REST and SOAP services can be discovered using their metadata. Taverna supports searching on the BioCatalogue and inclusion of a chosen web service through its workbench tool.

It is often necessary to do intermediate processing to be able to connect two services, where for example one produces the data that is required for the second but in a different format. Taverna provides a set of what they call "shim" services to cater for this need. Other types of services that are supported by Taverna are local, grid and cloud services, access to BioMart, R-Scripts and distributed command-line scripts.

The number of ready-to-use workflows have grown in recent years, which is ideal for researchers that need a starting point for their experiments. However this number is so large that finding the right pipeline for an analysis is not an easy task. For this reason MyExperiment was developed, it not only supports

Taverna workflows but also other systems such as Galaxy (discussed below). To run a workflow found in the MyExperiment repository in Taverna is as easy as copying the URL into the workbench importer, adjusting the parameters and pressing "run" [122].

### Galaxy

Starting as a project to integrate genomic sequences, their alignments and functional annotations, Galaxy has evolved rapidly in less than 10 years to the point of offering a complete web-based framework that aims to enable reproducibility, accessibility and transparency for computational biology experiments [35, 37].

First versions of Galaxy were written in a combination of C for the core components and Perl for the user interface, promising the possibility of adding new tools thanks to its architecture. Nowadays the project has been rewritten in Python and follows an open source strategy with over a hundred commits per month on its public repository (`https://bitbucket.org/galaxy/galaxy-central/overview`), and the extensibility promise has been fulfilled to the point of asserting that Galaxy supports any tool that can be run in the command line.

This feature marks the greatest difference between Galaxy and Taverna, because the latter has web services as its basic workflow unit while Galaxy is used to compose workflows using command-line tools.

Galaxy's main goal is to provide a tool where experiments can be reproduced easily and reliably; and Galaxy's authors consider that because it is in the nature of web services to be hosted by remote and probably unknown service providers, the reliability is compromised, and therefore web services are not natively included in Galaxy.

Despite the fact that the origin of Galaxy did not explicitly include the execution of workflows as its core functionality, Galaxy's detailed history of task executions has evolved and currently it offers all the advantages of a modern workflow execution suite.

The addition of metadata to both workflows and tools, and its publication by the means of Galaxy Pages makes the project easier to share. A well annotated workflow can be understood better when its documentation goes beyond the sequence of tools that have been connected, and includes the biological meaning of such executions.

There is an instance of Galaxy publicly hosted at `https://usegalaxy.org/` where a subset of features can be explored by anonymous users and the rest of the functionalities become available after free registration. This public tool includes hundreds of tools for getting data, processing it and visualizing it. If however, a project has particular requirements such as including non-public tools, Galaxy can be installed as a local server and the tools can be added to that instance [35, 37].

*Semantic integration*

The main methodology under semantic integration is to structure the data using semantic web standards (e.g. RDF, OWL) in order to make it "machine-readable" and be able to deduce meaningful associations. The conversion of the data into RDF files might not be a trivial endeavour, and it has similar problems to the ones mentioned above because it usually implies maintaining copies of the information in a separate format.

BioMOBY

BioMoby is one of the attempts to bring the promises of the semantic web into bioinformatics, proposing an architecture for the discovery and distribution of data though web services using multiple proposed standards of the World Wide Web Consortium (W3C) such as the Simple Object Access Protocol (SOAP). Its main goal is to provide access to biological data and services with a common format among the different sources [121].

The strategy of BioMoby is to define a minimalistic entity to describe the data in such a way that different types of data can use the same schema. This structure has been called the MOBY object, and is composed of three values: the MOBY object type (e,g. Sequence), a namespace identifier (e.g. Genbank/AC) and an accession number (e.g. AY070397.1). MOBy object types are defined using XML Schemas (XSD) reflecting a hierarchical relationship between them.

An important component of the BioMoby ecosystem is MOBY Central: a server that contains information not only about available services, but also their association with MOBY objects for input and output. With this information a MOBY client can suggest paths to follow depending on the current type of your data. Moreover, the intrinsic semantics of this approach is the ideal environment in which to create cohesive workflows.

Figure 1.1.5 shows the adaptation done by BioMOBY to the classic web services architecture: once all services have registered with MOBY central, a client can query it to find out which services can be used with the current MOBY Object. Besides the discovery feature, the central repository has the capability of generating a Web Service Description Language (WSDL) document that can be used with the corresponding service, where both input and output are MOBY Objects.

A more recent iteration of the development of MOBY was described in [113]: Moby-2. The objectives of the project have been extended and its focus on semantic web technologies is stronger: MOBY Central will take the form of a Resource Description Framework (RDF) triple store and will support the functionalities of a SPARQL endpoint. These changes were done with the objective of allowing semantic queries over multiple sources. The project was reported in [113] as a prototype, but

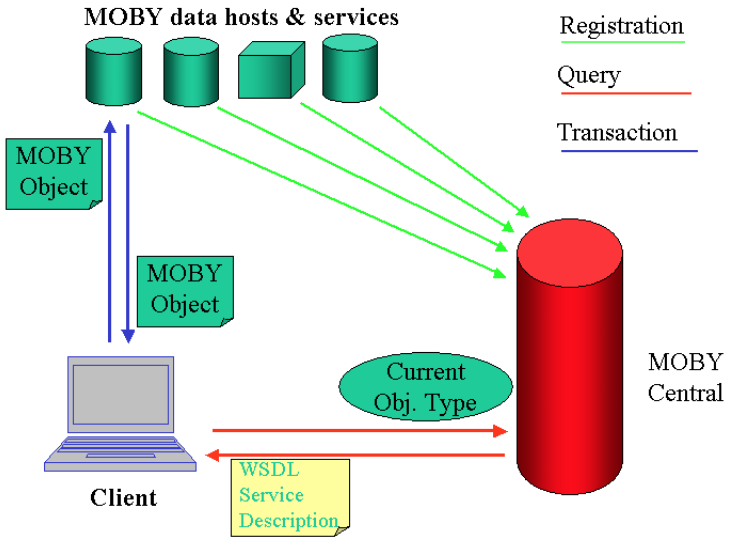


**Figure 1.1.5:** BioMoby Overview.

promised to contribute towards a distributed, machine-readable semantic web of life science data.

Bio2RDF

A simplified view of the Semantic Web can be defined as a network where the nodes represent any entity that can receive a name and the edges correspond to the characteristics used to describe the relationships between the nodes. The RDF format is an XML-based language that describes this relationships as a set called a "triple" that contains subject, predicate and object.

Bio2RDF uses RDF documents to be able to create a knowledge base that takes advantage of existing developments on the semantic web to build a mashup of data in the bioinformatics domain. [8].

One of the main objectives of Bio2RDF is to extract information from the most important bioinformatic databases, transform its content into RDF, and load it into a triple-store. It is within a triple-store that discovery of new knowledge occurs through recombination analysis of the loaded data. In this sense, Bio2RDF follows the approach of a data warehouse of semantic data.

As part of the project, a set of software tools that convert different datasets into RDF called "Rdfizers" were created. There is one rdfizer for each source, however they can be grouped in three types depending on the origin of the data: XML to RDF, SQL to RDF, and text file to RDF. For large scale resources such as UniProt or PubMed where programmatic access to the data is provided, the rdfizer works on-demand, which means that it only stores some of the data for cache purposes and the RDF files are created on the fly. Any other source gets copied into the centralised repository in order to offer quick responses.

The Bio2RDF approach can be summarised in three steps: (1) build a list of namespaces for data

providers, (2) analyse a data source to represent it in an RDF model and (3) develop rdfizer to convert the information. The resulting dataset is sorted in the triple-store in order to connect it all together.

Bio2RDF used an extended version of Sesame as the triple-store and on its 2nd release was changed to Virtuoso in order to offer better support to SPARQL, which is the most widely used query language for semantic web data. Bio2RDF makes use of other tools such as Protégé: an ontology editor, the Piggy Bank: a semantic browser for Firefox and Welkin: a RDF graph visualizer; all of which are well known projects in the semantic web community [8].

### SADI

The Semantic Automated Discovery and Integration (SADI) project has its roots in the lessons learned from BioMoby, particularly Moby-2. However, unlike BioMoby, SADI is not a data typing system. The goals of SADI are centred on the proposal of patterns and best practices of how to use web services connected through semantic web technologies to enable the creation of interoperable and integrative bioinformatics software. [120].

The biggest change introduced by SADI in respect to SOAP web services is to replace the defined languages for communication between the web services parts (e.g. WSDL, XML Schema,UDDI) with structured versions of semantic web languages: RDF and OWL. The authors go as far as to say that XML Schema is the problem causing the failure of most previous interoperability architectures.

A description of an interaction with a SADI service is as follows: A client requests the service description via HTTP GET, and the server responds with a document containing references to OWL classes describing input and output datatypes for the service. The client uses data formatted in RDF that follows the received description to submit an HTTP POST request, which is captured by the server, which in turn uses it to execute the service and generate a response in RDF format.

The adoption of HTTP methods for communication follows the positive reception of "RESTful" Architectures. SADI does not claim to follow a RESTful architecture, but it sees the potential of it and uses some of its principles. This is partially a response to the general dislike of the SOAP architecture within the bioinformatics community.

The use of a good ontology to connect services is seen in SADI as the key component to meaningful interoperability, where the interaction between servers is guided by biological knowledge and not only by technicalities such as format and availability [120].

As part of the project, software components have been developed in order to facilitate the adoption of the recommendations including plugins for Taverna and Protegé, and a prototype of all the components (i.e. Servers and clients) is available at `http://biordf.net/cardioSHARE/`.

BioPAX

The Biological PAthway eXchange (BioPAX ) is a community driven effort to develop a standard language to facilitate knowledge representation of biological pathways at molecular and cellular level in order to enable the systematic collection, distribution and integration of pathway data from heterogeneous sources [19].

BioPAX also takes input from the semantic web community. In this case OWL is used to define an ontology to describe pathway information, which can be used to interconnect the multiple resources in this domain. BioPax has, among others, been used to describe (1) metabolic pathways, following the abstraction: "enzyme, substrate, product"; (2) signalling pathways for biochemical reactions, binding, and catalysis events; (3) gene regulatory networks involving transcription and translation events and its control; (4) protein-protein interactions and protein-DNA interactions; and (5) genetic interactions i.e. when the phenotype of perturbing two genes is different from the expected known phenotype of the perturbation of each isolated gene.

The BioPAX ontology is the result of ongoing periodical workshops that involve the different stakeholders in the biological pathways field. Incremental versions of the agreement, also called levels, have been developed with the concept that newer levels can replace older ones. Currently the highest is level 3.

A tool set called Paxtools has been developed as part of the project. The main features of this software include an implementation of the specification as a software model, the support of OWL properties, a syntactic validator, transformation scripts between levels, import and export to other formats. Thanks to these features, Paxtools can and has been used as the framework to develop other tools [19].

*Wiki-based integration*

It is a cooperative effort where the community inputs information in an open and unstructured way, which can reach a highly reliable status as has been shown in the case of wikipedia. It is however completely dependent on the adoption of the community, and also given the unstructured nature of the data, is hard to manipulate for automatic analysis.

WikiPathways

Following the success of wikipedia, where any user can contribute to an article and the tasks of editing and curation are community based; WikiPathways has been developed with the objective of providing an open platform to deposit, share and curate biological knowledge in the form of pathway diagrams [55].

Biological pathways are representations of the compiled knowledge of biological units and their

relationships. Pathways are vital to understanding genes and proteins in terms of larger systems and processes of any organism. The challenges of gathering knowledge about biological pathways are particularly hard: (1) pathway information is not measurable and can't be obtained from a single experiment, (2) many different representations and methods have already been used, and (3) representations are usually saved as static images, which are far from ideal for computation and integration [85].

Looking to tackle these challenges and inspired by how science has been gaining a more open approach by means of open journals, public databases, data exchanges formats, ontologies and free software; WikiPathways provides a web based framework where the community can not only take information but also give back.

In WikiPathways, each pathway has a dedicated page that summarises the existing information around the specific biological mechanism including its diagram, description, links, related genes and proteins, and relevant literature references. The pathway is displayed using an interactive viewer that supports navigation and live highlighting. Registered users can also use the viewer to improve a pathway, and all the information can be exported in suitable formats such as the BioPAX standard.

A subset of the functions of the web-site can be programatically accessed via web services.

The metadata associated with the pathway serves the purpose of making it searchable, but most importantly makes it easy to integrate with other resources because it follows an ontology that as a side effect can be used to organise the created pathways in a hierarchical fashion.

Nonetheless it is clear to the authors that the tools and developments only assist in the community building process and its in the growth of the community itself that the future of WikiProteins is held [55].

### WikiGenes

In contrast to what its name suggests, WIkiGenes is not exclusively about genes; its scope goes beyond genes and aims to construct a knowledge base of biological information including chemical compounds, proteins, organisms, pathologies and of course genes.

Similarly to WikiPathways, WikiGenes applies a strategy based on the wiki model, however in [42] the author argues that given that the wiki model was not created taking into account the demands of the science ecosystem, it requires significant technical innovation. In particular, current wiki alternatives do not consider the current scientific publications paradigm, and the relevance of authorship.

The advantages of having a continuously updated article on each topic are obvious, however the effort from contributing scientists to reach this goal is quite considerable and the personal benefits of such efforts are not very clear. Traditional scientific publications recognise the efforts of a contributor by clearly stating its authorship, which in today's academic world might get reflected in employment, grants

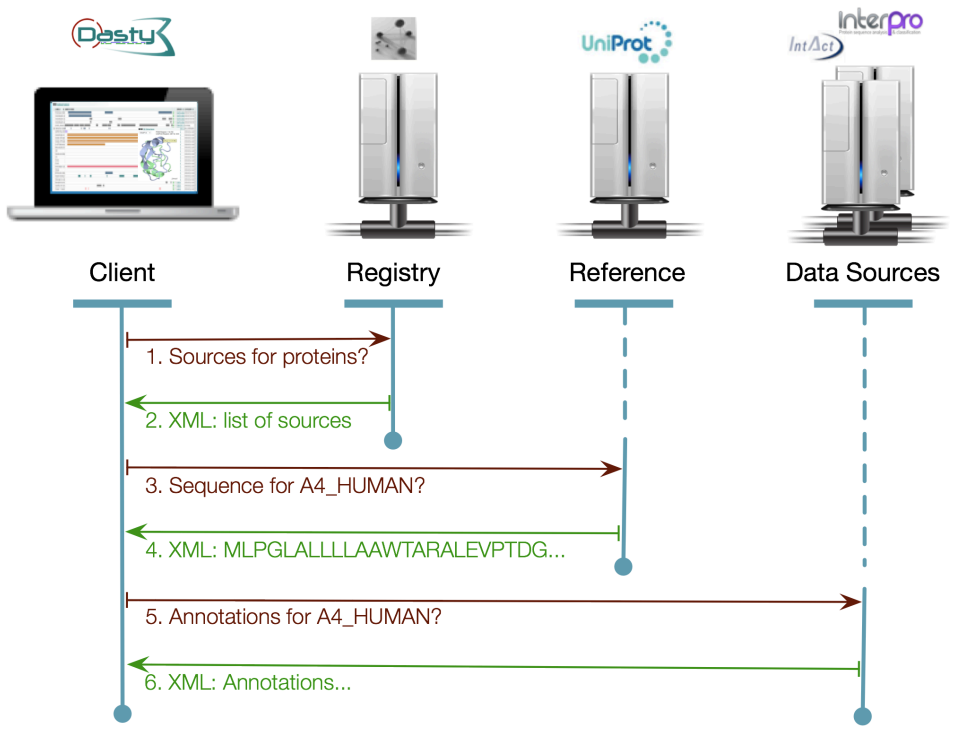


**Figure 1.1.6:** Flow of information in a standard query in the Distributed Annotating System.

and ultimately in the privilege of being a scientist.

WikiGenes proposes a system where every word of a document can be linked to its author in order to provide him/her with his/her due recognition. Such a document is reviewed by its readers as in the wiki model, however WikiGenes includes a reputation system that can be used to solve disagreements between authors and avoid vandalism.

More than a hundred thousand generated articles on several biomedical concepts have been included in WikiGenes. It is clear that the quality of these articles is not the best, but it serves as a starting point for interested authors [42].

### 1.1.2 The Distributed Annotation System

The Distributed Annotation System (DAS) [21] makes use of a widely-adopted standard communication protocol. It is motivated by the idea of maintaining a federated system; a logical association of independent sources distributed over multiple sites, which provide a single, integrated, coherent view of all resources in the federation. This architecture makes several distinct physical data sources appear as one logical data source to end-users.

A regular flow of information in DAS is shown in Figure 1.1.6. The DAS client requests information

about a protein that can be specified by its accession or identifier. The client then communicates with the DAS registry in order to retrieve a list of available sources providing information about that biological product. Once the client has retrieved this list, it proceeds to query the DAS reference source, i.e. a DAS source providing the sequence or structure of each molecule that it describes – UniProt in the case of proteins. The DAS reference source supplies not only the sequence but also meta-data such as the version. Thus clients can ascertain which retrieved annotations correspond to the original request. At this point, the client retrieves features, i.e. annotations, from the available DAS sources. These annotations may be applicable to specific subsections of the sequence (e.g. the location of active sites or observed peptides) or may be applicable to the entire sequence (e.g. related publications or taxonomy). Finally, the client organises and displays the annotations [93].

All these interactions follow an adaptation of the REST protocol for web services [88] .

*DAS Protocol*

The DAS specification consists of a set of rules which define a standard communication method between the different components of the system. DAS is Web-based and makes extensive use of three widely-adopted standards: the Unified Resource Locator URL, the HyperText Transfer Protocol HTTP and the eXtended Markup Language XML. All communication occurs through HTTP; the requests are URLs that specify the resource that the client is interested in, and the responses are both HTTP codes and XML documents. The details of what constitutes a valid URL, and the XML structure, are contained in the DAS specification.

By the time the first paper about DAS was published [21], the DAS protocol was version 1.01, and the main characteristics, such as the *features* and *dna* commands of DAS, were present in that version. From that point, several versions were released with minimal changes. These subsequent versions mostly just polished details to make the protocol stable and useful. The last official release of DAS was Version 1.53 on March 21 of 2002. This was the official version for several years, but in 2006 a new version appeared (version 1.53E) incorporating several new developments. These included an extension to serve new data types and an ontology for protein features [49]. The E in the version number is for Extended, which essentially describes the purpose of this version, because it keeps most of the features present in 1.53 but extends these to provide some new capabilities.

In November 2007, a project that aimed to define a completely new specification for the DAS protocol was concluded. The new specification was called DAS 2.0 (`http://biodas.org/documents/das2/das2_protocol.html`) and it contained a redefinition of the protocol for the capabilities that DAS had in its previous versions (1.0, 1.53). It also defined new features which allowed for the use of the protocol in a more extensive way. A controversial topic in the DAS community

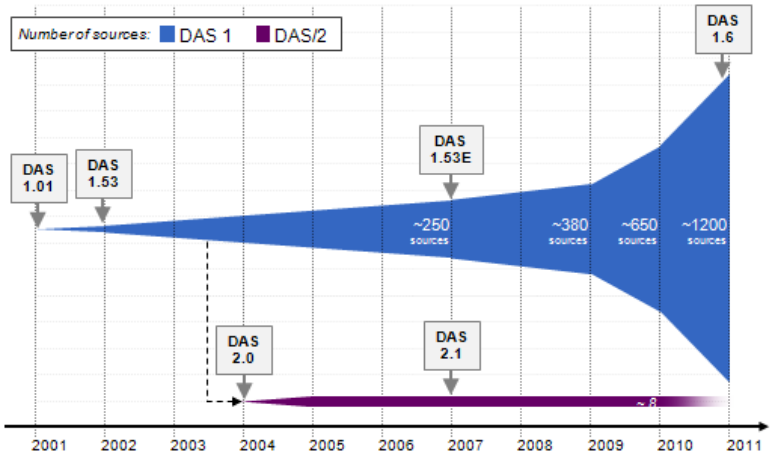


**Figure 1.1.7:** DAS Evolution.

was whether or not the DAS2.0 protocol should be adopted. This specification contains several improvements to the DAS protocol, but given the drastic changes in the format, amongst other reasons, most of the sources decided to continue using DAS1.53 or 1.53E. After the 2009 DAS workshop, it was generally agreed that most of the useful additional features that 2.0 provides would shortly be implemented in DAS 1.6E and its subsequent incarnations. As a result, DAS2.0 is now considered by many to be redundant. Figure 1.1.7 represents the evolution between the various version of DAS, and serves as a comparison between DAS 2.0 and 1.53E/1.6E in terms of number of sources, which is a good indicator of its adoption.

Version 1.53 of DAS defines the element *FEATURE* as the annotation itself, and it is contained in the element *SEGMENT* indicating that a feature annotates a specific segment, where a segment is a biological residue (or part of it), such as, proteins, genes, chromosomes, etc.

In the scope of proteins, annotations can indicate information about the structure (known formations of amino acids such as helices or sheets), interaction zones (with other proteins or regions of the same protein), phenotype (for example a known relation of part of the protein with a disease), etc.

An effort to group and organize all the types of annotations has been made: the Biosapiens Ontology contains, in a hierarchical structure, the types of annotations that can be used. As with any ontology, the information is not complete and periodic releases are made trying to establish a set of types as efficiently as possible. *TYPE* is probably the most important element included in a *FEATURE*. The use of the Ontology is highly recommended but is not mandatory, in order to comply with older releases.

The use of a second ontology (Evidence Code) is also recommended in the attribute *category* to express the method through which such an annotation was acquired, for instance by experiment, by *in-silico* analysis, etc.

Relevant information for proteins included in the element *FEATURE* is registered in:

- *id:* A source can not have two features with the same id.
- *label:* A human readable label for the feature
- *START* and *STOP*: Indicating the specific position to be annotated. If both are equal to zero, it means that the annotation applies to the whole segment (i.e. *Non-positional feature*)
- *LINK*: To indicate a URL where more information about this annotation can be found.
- *NOTE*: Space where the annotator can put any extra comment about the annotation.

Other elements and attributes are more oriented to other kinds of biological data, for instance the *ORIENTATION* element is useful for genes, to indicate if the annotation follows the direction 3' or 5', however proteins do not have an orientation.

DAS can be seen as having a *"Dumb server – Smart Client"* architecture where most of the hard work is executed in the client. Nonetheless, several independent projects have contributed to both clients and servers.

Under the DAS terminology, servers and sources represent two different concepts. A *DAS source* provides data for one *Coordinate system* (`http://www.dasregistry.org/help_coordsys.jsp`), i.e. a unique 4-tuple *(Authority, Version, Type, Organism)*, e.g. (Ensmbl, 51, Chromosome, Homo Sapiens). On the other hand a *DAS server* is the software that facilitates the publishing of DAS sources.

Each DAS source can support several capabilities, which means it is able to respond to an HTTP request that gets interpreted as a DAS command (e.g. sequence, features) with a document that follows the specification. DAS servers are pieces of software that implement the common tasks of this process, for example handling the HTTP requests, providing a logical model for DAS or encoding the model into a document.

The two most representative implementations of a DAS server are Pro-server [26] and MyDas [95]. Both have been updated to support the latest version of the specification (i.e. 1.6E) and their feature sets are similar. Probably the biggest factor in choosing between these two implementations is the preference for a particular programming language: Perl for Pro-server and Java for MyDas. An extended description of MyDas can be found in the section 2.1, including the contributions to MyDas as part of this PhD project.

On the other side of the spectrum, the DAS clients have the task of providing a unified view of multiple sources. Most of the DAS clients centered their efforts on a particular DAS type, such as proteins (e.g. DASher [72]), chromosome (e.g. Ensembl viewer [28]) and genomes (e.g. Dalliance [22]), where others such SPICE provide a way to navigate between multiple DAS domains. For instance, it is possible

in SPICE to start on a chromosome view, zoom-in into a gene region, select the expressed protein of the gene and visualize its 3D structure, all in the same Java Web-start window [87].

Dasty2 is a Web client that also supports the interaction between protein data and its 3D structure, but its current version doesn't support direct manipulation of genomic data [50]. A refactoring of Dasty was executed during 2010 and is explained in detail in section 2.3.1, including our contribution to the effort of developing Dasty3 [114].

### 1.1.3 Discussion

Besides the primary data from *in-vitro* experiments, there are hundreds of secondary sources consolidating data that results from *in-silico* analysis. All the projects mentioned contribute in different ways to the creation of a pool of knowledge where both primary and secondary sources are available to the researchers.

Some of these projects started when a problem was detected while trying to compile the generated data of a particular community (e.g. BioPAX); while others have studied an existing technology such as data warehouses, web services or semantic web and proposed adaptations to it for bioinformatics needs (e.g. Biowarehouse, BioMoby). Some approaches are on the protocol and specification level (e.g. SADI) while others take existing specifications and generate the tools to facilitate their implementations (e.g. BioRDF). There are projects that take existing software and adapt it to bioinformatics (e.g. WikiPathways) and others develop completely new software (e.g BioMart, Taverna). Some focus on the integration of the data (e.g. BioMart) and others on the interconnections between components (e.g. Galaxy). Some focuses on the tools (e.g. SRS) and others on the results and their publications (e.g. WikiGenes).

Despite their origin, methods, technologies or approaches; all of these projects have in common the need for a strong community that supports its development, maintenance and use. We chose DAS as the base technology on which our contributions will be focused because when we started the project, it had a growing community as seen in figure 1.1.7. It was based on existing and consolidated technologies such as the HTTP protocol and REST services, and it had the support of big entities and projects, such as the EBI and Ensembl. A description of the specifications and software components developed as part of this PhD project can be found in chapter 2 of this document.

## 1.2 Visualization

The field of visualization aims to represent data in such a way that non evident features become visible. The developed techniques to achieve this objective vary from simple ones (e.g. histograms) to the very elaborate (e.g. environments only visible using 3D virtual reality rooms).

The uses of visualization techniques are as diverse as the fields in which they are used, from weather forecasts in the news to the analysis of the captured data in the Large Hadron Collider. In our field of interest, bioinformatics, the use of visualization methods is also abundant, and sub-fields such as genomics, proteomics, population variance, etc. have plenty of examples where different techniques have been implemented with the purpose of making sense of biological data via visual representations.

The most representative visualization techniques used in bioinformatics can be categorised into three groups: charts, networks and hierarchies. Charts are graphical representations where n-dimensional data is mapped or aggregated into a space, for example scatter plots, bar graphs, pie charts, etc. Networks are structures where some components are connected to an arbitrary number of other components, normally using a node-link metaphor where the components are represented by nodes and the connections are lines between them. Lastly, hierarchies are another type of connection that shows when an item is part of another, or using graph theory terminology: a node can be the parent of another. The diverse representations of trees can be used for this type of data, for example, node-link trees and space filling diagrams [118].

Figure 1.2.1 shows a a comparison of several libraries that have been used in bioinformatics research [118]. The libraries in (a) include the capability to represent charts, while (b) and (c) are show similar graphics for libraries that support network and hierarchical representations respectively. It is worth noting that several libraries are present in all three graphics because they have been developed without a specific domain in mind so that they can be useful in a variety of applications.

A recurrent challenge when visualizing data is to match the most appropriate combination of techniques to the nature of the data, optimising the features of the visualization (e.g. location, colour, size or shape) in order to highlight a biological characteristic (e.g. genomic position, functional class, expression level or organism). Or as stated in [32] "*The challenge is to create clear, meaningful and integrated visualizations that give biological insight, without being overwhelmed by the intrinsic complexity of the data*".

A variety of projects in different subdomains of bioinformatics have been presenting alternatives according to the needs of each case. The sections below describe some of the most illustrative projects, grouped by some well known bioinformatics domains.

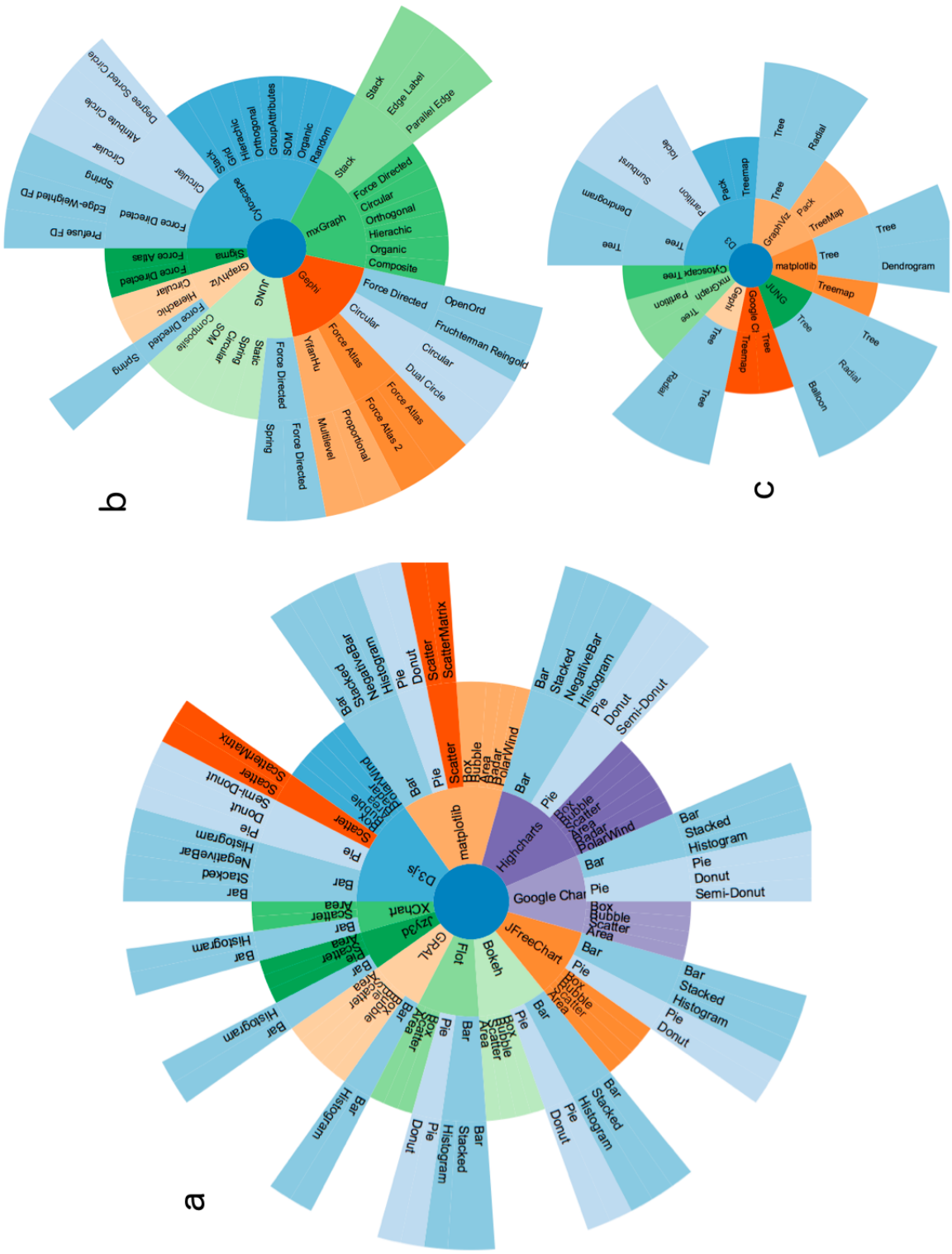


**Figure 1.2.1:** Comparison of features of selected visualization libraries.

### 1.2.1 Genomics

Genomics is arguably the subfield that produces the most abundant, basic and yet relevant type of data in molecular biology research. Advances in the methods for extracting DNA, RNA and other biological products have grown dramatically in recent years. For example, sequencing projects have progressed from taking decades to gather the approximately 3.3 billion base-pairs *pb* of the human genome project, to the ability to extract over a billion short reads (i.e. individual sequences of approximately 100bp) in days or even hours with some of the new technologies known as Next Generation Sequencing (NGS)

This comparison may however be unfair because the number of reads produced in NGS still requires a large amount of processing in order to assemble a genome because it is more challenging to align NGS short reads than the reads of the first generation sequencing, however this extra processing time is in the order of days, not the years required previously. Moreover, none of the NGS technologies would have been possible without the findings of the human genome project. Nonetheless the comparison still serves to reflect the rapid growth in the methods of sequencing DNA.

Visualizations have played an important role in the different stages of genomic analysis. In [76], the authors have identified three tasks in genomic research where visualizations have been widely used: (i) supporting the sequencing process, (ii) browsing annotations in the context of a reference genome, and (iii) comparing sequences from different organisms and/or individuals.

*Sequencing Process*

Despite all the progress in sequencing techniques, the process is not perfect and requires visual inspection in order to interpret and validate automated outputs to compose sequences, chromosomes and ultimately genomes. It is a common strategy to associate quality scores (QS) to each of the nucleic bases that result from a sequencing experiment. The QS is proportional to the number of agreeing reads that cover a base. For example the QS on a given position X would be low if half the reads covering X indicate a G and the rest point to a T. It would be similarly low if only few reads cover that area or if the bases are not clearly distinguished in the reads. In contrast, if many reads are covering an area and all of them agree to assign A at X, the score in that position should be high.

Several tools have been developed which take advantage of this information to align the reads and represent the scores in multiple ways. For example histograms or heat maps ease the task of identifying regions of low coverture and expose errors in the automatic consensus sequence. This type of tool is highly dependent on the technology used. For instance, there are no raw read traces(only images) in some of the NGS techniques, and therefore a detailed alignment is prohibited because of the elevated computational cost.

Some of the read alignment viewer tools go beyond the display functionalities to allow editing of the

assembly in order to complement the automatic outcome and to annotate relevant part on the sequence (e.g. genes, promotors, etc).

HawkEye was one of the pioneering tools that offered some of the features discussed above, and mainly focused assisting in the detection and correction of assembly errors for whole-genome shotgun projects. [100].

This stand-alone tool offers a Top-Down strategy to analyse an automatically assembled genome. Figure 1.2.2 presents the three main views included in Hawkeye:

**(a) The launch pad** is the first screen presented to the user when a draft genome has been loaded. It acts as a global overview by displaying summary assembly statistics in the form of two N-Plots (i.e. A bar graph where each bar is a container), its height represents the length of the contain in pb, and the width its length in percentage of the genome size.

**(b) The Scaffold View** represents the current scaffold as a linear ordering of connected contigs, with the assembly features displayed below. The first two tracks are heat map plots to easily visualize the insert and read depth of coverage. This view allows zooming and panning in order to navigate through the assembly.

**(c) The Contig View** follows the same design as the scaffold view by representing the consensus on top and the composing items aligned below, however in this case the level of detail can be expanded to show the nucleotide bases along with their quality scores and the chromatogram traces when available.

The formats to store read alignment information have been influenced by the technical changes imposed by the new technologies associated with NGS. The SAM format was created as part of the 1000 genomes project as an attempt to provide a generic alignment format that supports single and paired reads. A SAM file can then been converted to BAM, which is a binary lossless compressed file that can be indexed to offer rapid access to a specific position of the alignment [64].

A set of tools called SAMtools was developed by the same team in order to help with the basic operations required to manipulate a SAM file. It includes a Text Alignment Viewer, which despite being very basic and command line based, is of great help to researchers because its simplicity results in high performance, which makes the navigation of full genomes very fast.

Similar formats to SAM have been developed around the NGS technologies in order to store different types of data, for instance, BigBed and BigWig, which are the Big Binary Indexed (BBI) versions of the BED and WIG formats. BED files are used for tables with a varying number of fields, where each line contains the fields for one record separated by a white space. WIG files are used to associate a double

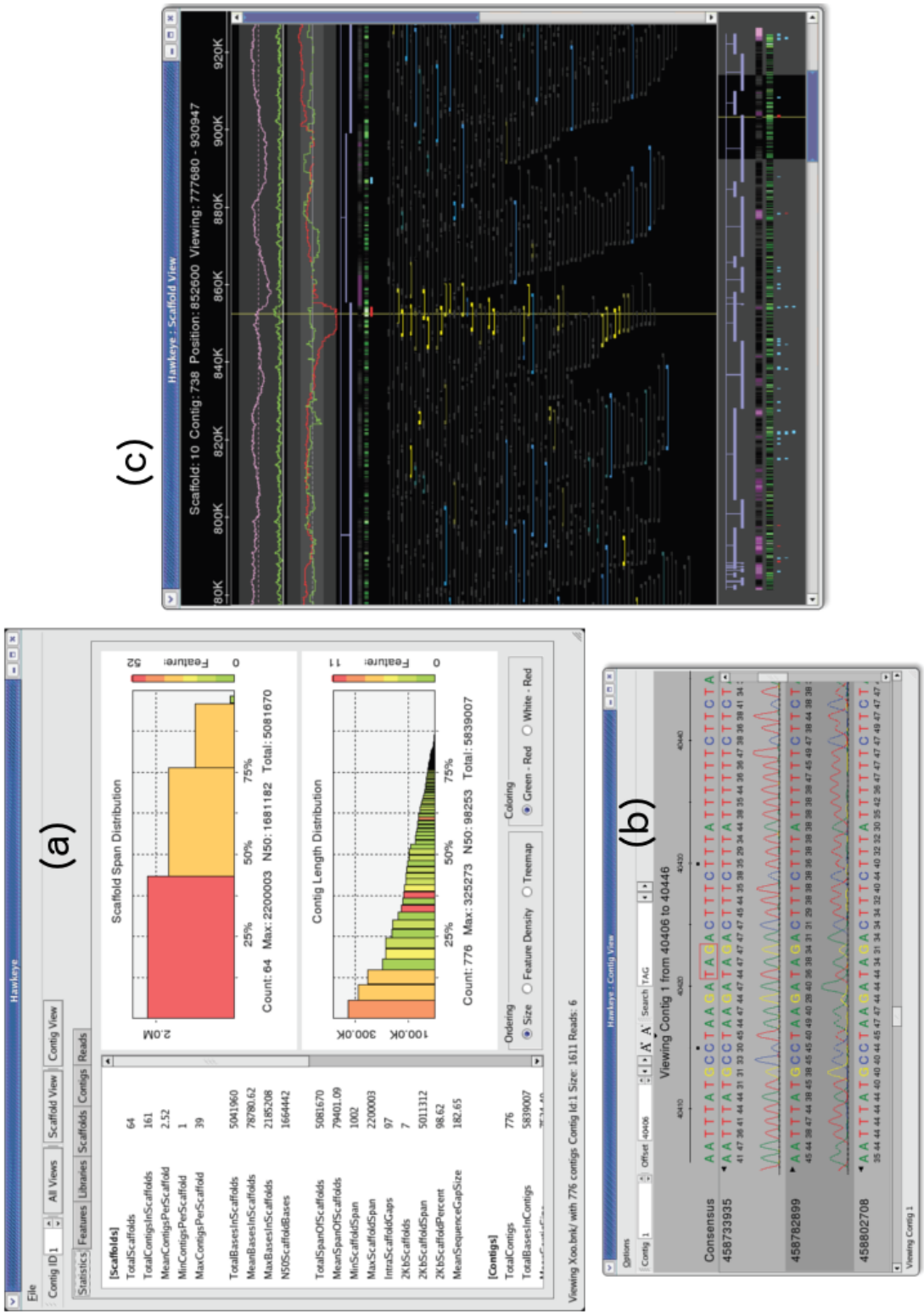


**Figure 1.2.2:** Snapshots of the available views in the Hawkeye tool: (a) Launch Pad, (b) Scaffold View and (c) Contig View.

value to each base. This format is designed to compress information when the same value is assigned to a large section of the sequence[56].

The Integrative Genomics Viewer IGV is one of the tools that takes advantage of these new formats to enable real time exploration of large-scale genomic data-sets at different resolution scales [89]. Data sets of both basic aligned read sets as well as deviated results can be loaded into IGV from local and remote sources. IGV can then be used to detect and correct assembly errors such as misalignments in repeat regions. IGV also supports the display of annotations files in context of the genome, making it suitable for the category (ii) defined at the beginning of this section.

A description of other tools that support the process of sequencing a genome (category (i)) can be found in [76]. Once the genome is ready and has been through a finishing process it is usually shared and other scientists can use it in context with their own data. It is at this stage where the tools of category (ii) i.e. browsing a completed genome with its annotations; are of great help.

*Genome Browsing*

A famous principle in visualization is known as the Shneiderman mantra: *overview, zoom, filter, details-on-demand* [104]. Most genome browsers use a similar layout that follows the mantra quite closely, adapting it to the molecular biology behind the data.

The *overview* is usually a graphical representation of the karyotype, which is the set of chromosomes displaying its bands. The bands have been experimentally defined using cytogenetics techniques to identify parts of the chromosome, and are used as visual markers for different chromosomic regions.

The user can *zoom* into a chromosome band, or can select a region of interest by interactively marking an area, or by explicitly introducing the coordinates. Once a region of interest (ROI) is selected, a multi-row visualization is displayed, where the first row represents the ROI. Some browsers use colour coding for this row to show the limits of the bands or other high level features of the region.

Each subsequent row is called a track and displays a different set of features drawn proportional to the selected region. For instance, it is common to include a track displaying the annotated genes in the ROI, using boxes to represent exons, and lines for introns. Other symbols can be used to represent different features, e.g. arrows to define if the gene gets translated in the direction 5'UTR or the other way around, or colours can be used to represent the functional class of the gene, etc.

Most genome browser have a wide list of sources that provide different types of information such as transcription factors, single nucleotide polymorphism SNP, details about the sequencing source (e.g. contains, reads), proteins, supporting evidence, etc. The tracks to be included in the browser are then selected and *filtered* by the user.

Finally a feature of interest can be selected to obtain *details-on-demand*, for instance, to get link to a

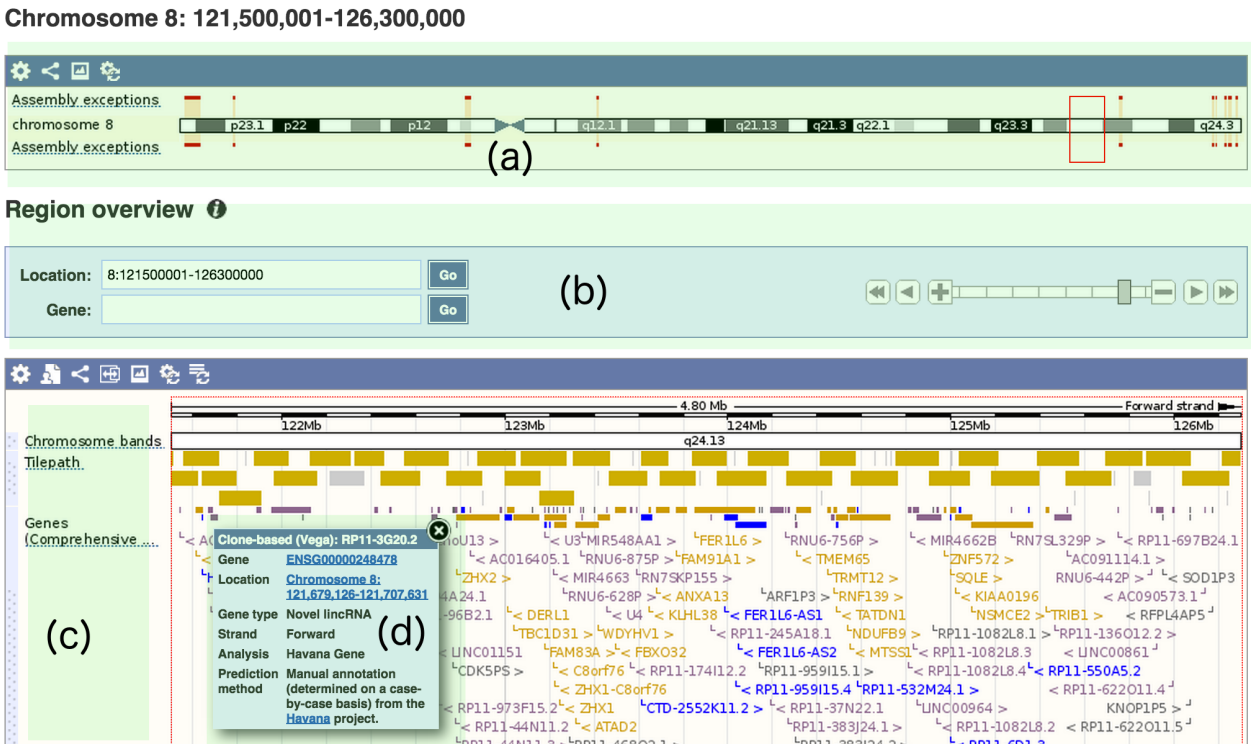


**Figure 1.2.3:** Snapshot of the Emsembl genome browser highlighting the components that support the implementation of Shneiderman mantra: overview(a), zoom(b), filter(c), details-on-demand(d).

website that contains the information of the protein that gets produced by the selected gene, or specific data on the allele frequency of a SNP and its associated phenotype.

Figure 1.2.3 shows some of the widgets used by the Ensembl genome browser that follow the Shneiderman mantra. Region (a) displays a selected chromosome showing the overview of the data, in (b) it is possible to select the zoom level and the coordinates to mark the ROI; (c) shows the name of the sources that have been selected for the current graphic, and (d) is a pop-up window displayed when the user clicks on one of the features of the graphic.

The Ensemble website (`http://www.ensembl.org/`) aggregates all the information that has been integrated through the Ensemble project. The dataset has grown from having partial information in 1999 for one species: *Homo Sapiens*; to include full support to 69 species by the time of the release of version 77 in October 2014.

From the beginning, the objective of the Ensembl genome viewer was to support multiple species with a single software installation. The challenge goes further than the amount of data generated by species, because there are many differences in the data from one species to another [108]. The project has clearly been successful, not only judging by the number of species included, but more importantly because of the

size of its community, reflected by more than 1500 user queries assisted by their helpdesk team in 2014 [17].

Besides the sources provided by Ensembl, it is possible to also include external sources by using several protocols. For example, it is possible to include a URL to several NGS files (e.g. BAM, BigBed, etc.), or to point to an existing data source that uses the DAS protocol (discussed in section 1.1.2).With this information, Ensembl will query that source to get information on the ROI and display it in locations views, such as Region in detail, chromosome or karyotype. Moreover, Ensemble is in itself a DAS source, which makes it easy to include its data in other resources [107].

Similar efforts to Ensembl have been addressed by two other big organizations: the NCBI Map Viewer [1] and the UCSC Genome Browser [90]. Both projects provide easy access to the centralised repositories of each institution, and allow third party data to be displayed in context with their own. Moreover, they collaborate with each other, and information from one source is available for display on the other. A comparison between these three genome browsers can be found in [31].

These traditional genome browsers share the same architecture, where both data and service are located on the server side: when a new request arrives, the data is collected from local or third-party data sources, and then an image is created by the server, which is ultimately transmitted to the client; HTML link-maps are also generated to include interactive links over the image. In this way the client only requires the ability to display images and the use of the *map* HTML element, which has been part of the HTML specification since early versions, and therefore is widely supported.

Unfortunately by using this approach, it is necessary to re-generate a completely new image with most user interactions, for example, by moving the ROI by a few bases, or by hiding a data source. Some strategies have been developed to improve the usability of these applications, for instance using cache memory in the server to accelerate the access to remote sources, or pre-generate the most likely required images (e.g. left and right of the current region). Nonetheless, factors such as the number of sources or network delays can dramatically affect the smoothness of the navigation in a system of this type.

Recent web technologies permit the decentralisation of the data, increasing the workload on the client and therefore liberating the server from most of the display related tasks. According to this model, the web genome browser is an application that mainly runs on the client, requesting data on-demand and generating/manipulating the graphic accordingly. If, for example, the user drags the display area to the right, making a few more bases visible, the client only requests the information of the new area, and alters the graphic dynamically. This reduces the computational cost of creating a full image, but most importantly isolates all the "look and feel" tasks from the server, allowing the server to focus on serving data, and consequently to attend to more and bigger requests.

JBrowse is an open source application that follows this approach implementing a robust client in

JavaScript with the support of a server side developed in Perl for the preprocessing of the data. JBrowse uses a combination of standard HTML 'div' and 'canvas' elements to represent the genomic features. For instance in high level views of data, 'divs' are used to create histograms to show the amount of features in an area. As in the case of other genome browsers, JBrowse supports the inclusion of third-party data using NGS files. For example, BigWig data is represented either as heat maps or histograms using 'canvas' elements [62].

There is another strategy for genome browsing, which takes advantage of the features of what is also known as Rich Internet Applications RIA. This preprocesses the information to create small images called tiles that can be used to compose a view, similar to the experience introduced by Google maps; in this way there is no need to refresh a whole page if there are minor changes in the selected area.

However, in [105] the authors of JBrowse present a benchmarking experiment, where the two approaches are tested. The results of the experiment show better response times in the case where they used HTML elements. Besides the performance, the 'tiles' approach requires a lot more storage space in order to have all the tiles pre-calculated, moreover, the approach complicates the desirable features of including third-party sources because it might need to pre-process the entire source to create the tiles.

A similar project to JBrowse is Dalliance, a Web-based Genome Viewer developed using the HTML5 capabilities to manipulate SVG elements. It also requires a modern web browser (Firefox 3.6+, Chrome 5+, Safari+). Dalliance takes advantage of the DAS architecture in order to allow the users to put custom data in context of reference genomes. As with other genome browsers mentioned above, it supports the interaction with BAM files and other NGS related formats [22].

Similarly MyKarioView uses DAS in order to include other sources, this tool places an special emphasis on the display of personal genome data in context with well known data sources.

It is important to highlight that the purpose of genome browsers is to simplify the task of generating hypothesis based in the aggregation of genomic data. Each data source can contain errors, and the researcher should be sceptical of any conclusion based on a single observation. Genome browsers allow the graphical display of several sources, and in this way make it easier to detect anomalies, errors or special genomic conditions. Despite how evident a hypothesis is represented in a visualization, the scientist must have access to the original data in order to evaluate any theory [12].

*Comparative Genomics*

The last of the categories (iii) mentioned at the beginning of this section, refers to the group of tools used to compare genomes in between individuals and/or species.

With the constantly growing number of complete genomes, it is a logical step in research to start looking for high level similarities between them and to use the discovered knowledge for one species in

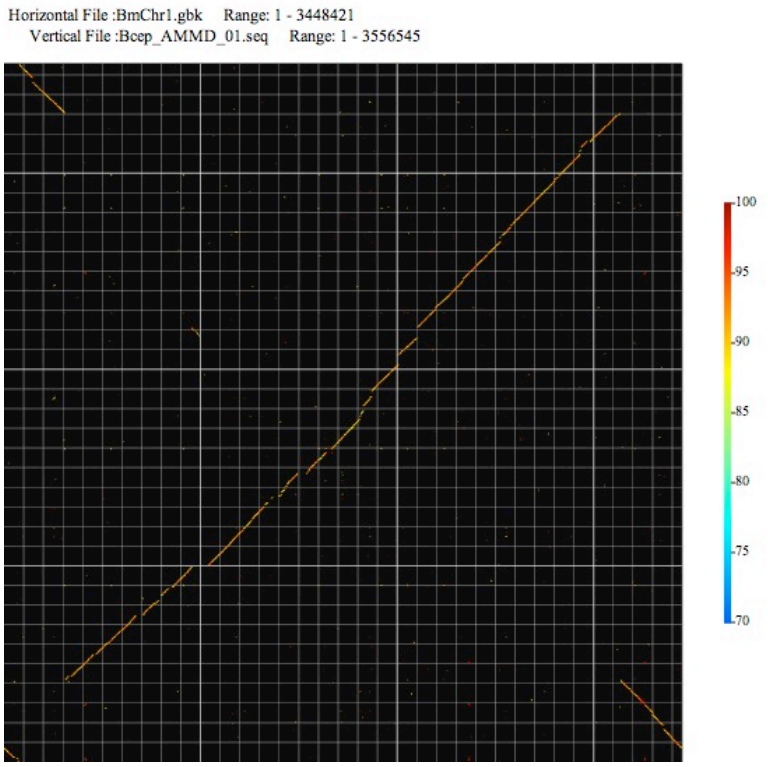


**Figure 1.2.4:** Dot plot comparison created with GenomeMatcher

the research of others. This field, known as comparative genomics, tries to identify functional elements common in multiple species and their evolutionary modifications. It is also common to use these techniques to assist in the assembly and finishing of new genomes.

The estimation of the conservation of the chromosomal location of multiple genes among several species is known as synteny. Its relevance lies in the assumption that a sequence of genes in the same location in two species is an indication of a common ancestor. A "dot plot" graphic (see Figure 1.2.4) is commonly used to represent the synteny between two genomes. Each of the two dimensional axes of the plot represent one of the genomes, and dots in the graph are the positions where a common gene is located. Connected dots that show a 45 degree line are the indication of a high syntenic region. Tools such Vista [30] and GenomeMatcher [80] include implementations of dot plots among with other ways to compare genomes.

As an alternative to dot plots, some tools offer a technique to display multiple genome alignments and mark corresponding areas in between by colour-shadowing. For example, Figure 1.2.5 shows an example generated using GBrowse_syn [71] where multiple genomes from the WormBase database have been

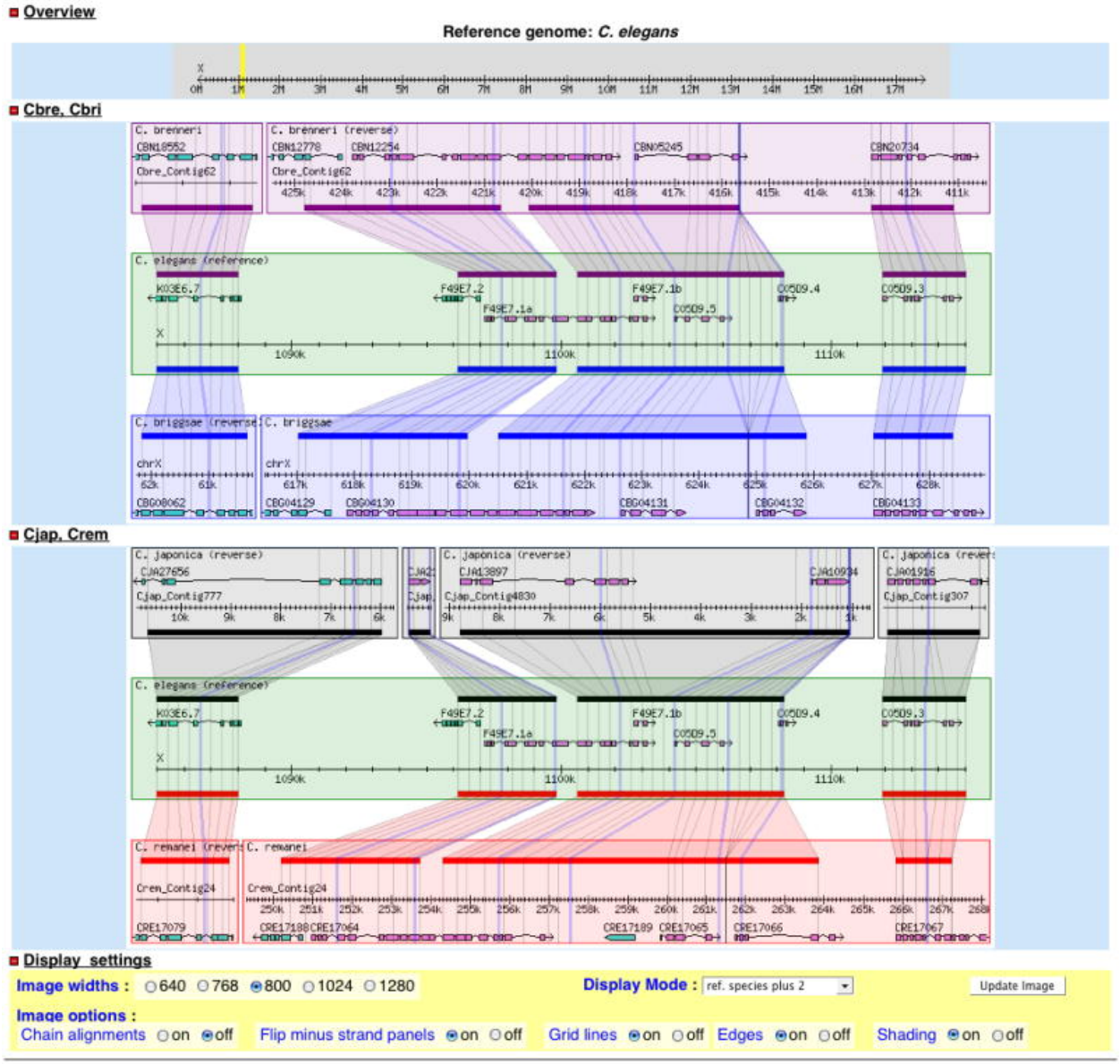


**Figure 1.2.5:** A five species whole genome DNA sequence alignment comparison from WormBase using GBrowse_syn.

aligned and a section is displayed to show the conservation of certain genes.

Besides the comparison of genomes between species, the progress of sequencing techniques permits the analysis of multiple individual's genomes in order to study their variation, and, in this way to try to discover the genotype that causes phenotypes of interest, such as diseases. The largest international effort in this area to date is known as the 1000 Genomes project, which at the time of their latest publication [14] includes the genomes of 1092 individuals from 14 populations, and discovered and genotyped 38 million SNPs, 1.4 million indels and 14000 large deletions. All of which is vital information for genomic studies of human health.

An adaptation of the Ensembl browser has been developed in order to provide a way to visually browse through the resulting information from the 1000 Genomes project. Multiple tracks including aggregates and results from the study have been added to the browser and new functionalities to export and visualise variation data are now part of this tool.

### 1.2.2 Proteomics

At the central dogma of biology, DNA (deoxyribonucleic acid) encodes genes, which get transcribed into RNA (ribonucleic acid), which in turn, gets translated into proteins. Each of these is represented by a sequence of either nucleotides (DNA, RNA) or amino acids (Proteins). In the previous section we discussed the visualization tools that assist in the process of sequencing DNA in order to get a full genome of a species, where positional annotations can be included, shared and explored in the context of one or multiple organisms.

It is because of the central dogma that annotations about genes are among the most used and useful features to know in a DNA sequence. The direct relationship between a gene and a protein is the key to a large part of molecular biology research. For example, many diseases are caused because there are not enough (or too many) proteins to execute a biological function, or because there are proteins but they are malformed. In any either of those cases the source of the problem can sometimes be traced back to the gene that generated that protein, and it is often found that a mutation has appeared and a section of nucleotides have been inserted/deleted (indels).

It is one of the goals of proteomics, at least in the clinical context, to deliver markers for disease prognosis, state and treatment outcome and targets for disease treatment [73]. The Online Mendelian Inheritance in Man (OMIM) resource is a catalogue of human genes and genetic disorders [3]. In OMIM it is possible to find what mutation in a gene (genotype) is associated with a disease (phenotype). This information has been compiled from peer reviewed publications that represent the current knowledge in the area from several communities (clinicians, molecular biologists and genome scientists). This, of course is a work in continuous progress and despite the enormous contribution of resources such

OMIM, it is not by any means a finished task.

Once a good approximation of the genome of an organism has been identified, deciphering its proteome (i.e. the total protein complement of a cell, organ or even an organism) is the next logical step to follow, especially considering that the number of identified genes during the human genome project ( 30000) is far too small to explain the complexity of human biology [81].

This is due (among other reasons) to gene and protein splicing and post-translational modifications PMS, which then makes it impossible to completely deduce the proteome from the genome. Therefore, approaches that start at protein identification are necessary in order to study proteins that otherwise wouldn't be detected. Besides, the discovery of genes by computational means is limited and proteomics techniques offer an alternative solution.

Multiple experiments have been developed in order to address this challenge. For example: mass spectrometry (MS) helps to identify which peptides and ultimately which proteins are present in a given sample; 2D gel electrophoresis and microarrays analysis can give insights in the proportions of proteins that have been expressed in certain conditions (e.g. an specific tissue, a sick patient, a health person(control), etc.) and similar data can be obtained with alternative uses of NGS technologies such RNASeq; crystallography and modelling software have been used to find the final shape of the protein.

Visualization has shown it's potential by helping in the understanding and analysis of the outcomes of such experiments. Below we will show some cases grouped by some of the most frequently used proteomics techniques, where visualization methods have been used to support the analysis.

*Gel Based Proteomics*

Two-dimensional gel electrophoresis 2DE is still the most widely used approach in top-down proteomic studies, mostly due to its efficiency at separating proteins in complex mixtures. Figure 1.2.6 presents the usual steps involved in an 2DE experiment that separates the proteins of interest for a particular hypothesis, which can be then identified with, for example, MS techniques [91].

The four first stages in this flow describe the preparation and execution of the experiment itself, which results on a set of gels (one per sample) where proteins are separated according to molecular size and isoelectric point. Conglomerations of proteins are then exposed as darker spots. In order to analyse the relevance of the different proteins to the hypothesis of the particular study, the gels need to be digitalised and processed, quantifying the protein concentration on each of the detected spots (alternatively some techniques quantify per pixel). Multiple techniques of image processing are applied in order to clean and extract the data, then normalisation and stabilisation is executed. The output of this process is a matrix in which every row represents a sample, and every column represents a spot across all gels. Values in the

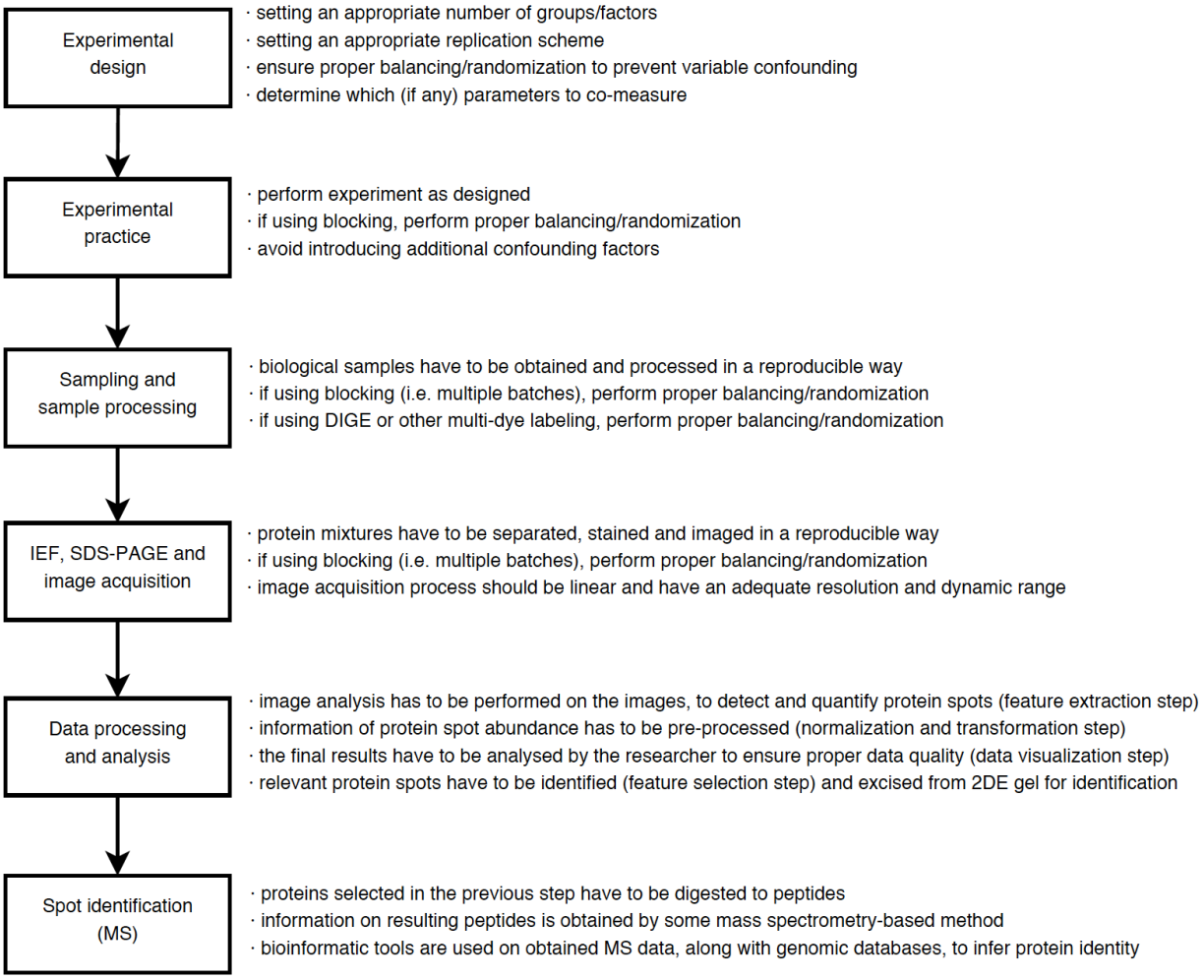


**Figure 1.2.6:** High level overview of a typical gel based proteomics workflow.

matrix are then the level of concentration on an spot of a given sample.

There are different protocols for gel-based experiments to try to overcome the different limitations of this technique (i.e. low resolution, low dynamic range, bias agains categories and low reproducibility), and Differential Imaging Gel Electrophoresis DIGE, has become one of the best alternatives. The main difference of DIGE to other protocols is that control and case samples are tagged with different fluorescent dyes, which facilitates the spot detection and its posterior quantification [81].

In most of the protocols, each of the gels can have other variables associated with them, for instance the tissue that it belongs to, information about the donor (e.g. sex, age, weight) and data about the hypothesis (e.g. healthy/infected). The relevance of a spot in the context of the study hypothesis and considering all the mentioned variables, is used to filter the required protein identification experiments, and ultimately reduce time and costs to prove/disprove the hypothesis. It is in this step where statistical and visualization techniques are of great assistance to the researcher.

The problem is that common visualization techniques (e.g. box plots, scatter plots) are usually not enough, because the number of variables(spots) to include is too high(in the order of hundreds), and therefore multivariable techniques that combine statistical methods and visualization tools are required at this stage.

Principal Components Analysis PCA has become the preferred technique in 2DE, because it offers a good and unbiased view of a dataset along the subspace where most variation occurs. PCA looks for a 2D or 3D representation of the data where the axes are the principal components PC (hence its name) that most highly contribute to determine the variance of the data. A PC is not a single variable but rather a projection of the combination of several, and the result of a PCA gives the contribution of each of the variables for each PC.

Figure 1.2.7 serves as an example of the visualization of the results of a PCA on a data set in which two treatments ("A" and "B") were compared with biological samples being taken at three time-points: 0h, 6h and 48h; colour-coded as light grey, dark grey and black respectively. There is a clear division of two groups. In a 2DE experiment, it is common to select the spots to analyse further by studying the variable contribution on each PC.

PCA uses the euclidean distance when calculating the projections of the variable contributions on each PC. This strategy has shown a good balance of outcome/performance for multidimensional data, however when the number of dimensions is very high it can be misleading, and then other techniques are necessary, e.g. Independent Component Analysis ICA, Partial Least Squares PLS, Metrical multi-Dimensional Scaling MDS, Non-Metrical multi-Dimensional Scaling NMDS, clustering methods and Self-Organised Maps SOM. A discussion about these methods in 2DE can be found in [91].

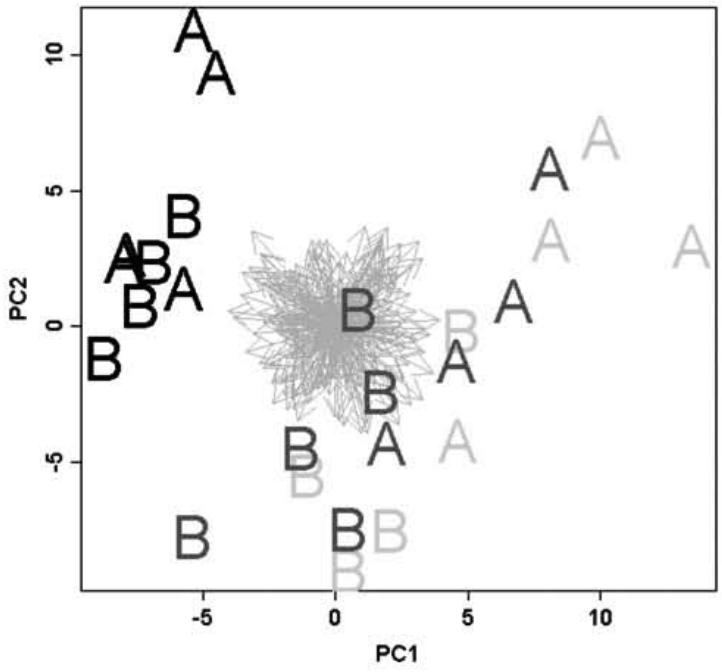


**Figure 1.2.7:** A biplot displaying PCA results

*Mass Spectrometry Techniques*

Proteomics based on mass spectrometry is about identifying, quantifying and characterising as many proteins as possible in a single experiment . Probably the tipping point for the usage of MS techniques was the definition of the tandem mass spectrometry (also known as MS/MS) because it allows the identification of a large number of proteins using high throughput techniques .

A MS experiment it is commonly the the next step after getting the results of a separation process. In the case of 2DE, the spots are cut out of the gel, and the proteins are digested into shorter peptides by an enzyme and then separated in order to reduce the sample complexity and to get better performance out of the mass spectrometer.

The workflow to analyse the MS/MS results is described in figure 1.2.8. The input is the row data generated by the mass spectrometer, whose format is usually native to the machine, and therefore the first step is to convert the files into more standard formats.

The mass spectra datum of each peptide is used to try to identify it which usually involves searching for similar mass spectra of known (or hypothetical) peptides; followed by a validation step, and then the inference of the target proteins is calculated

The subsequent stages in the workflow are dependant on the nature of the experiment (e.g. requires quantification because it is a comparison of control v.s. sample) or the setup of the laboratory (e.g. it uses

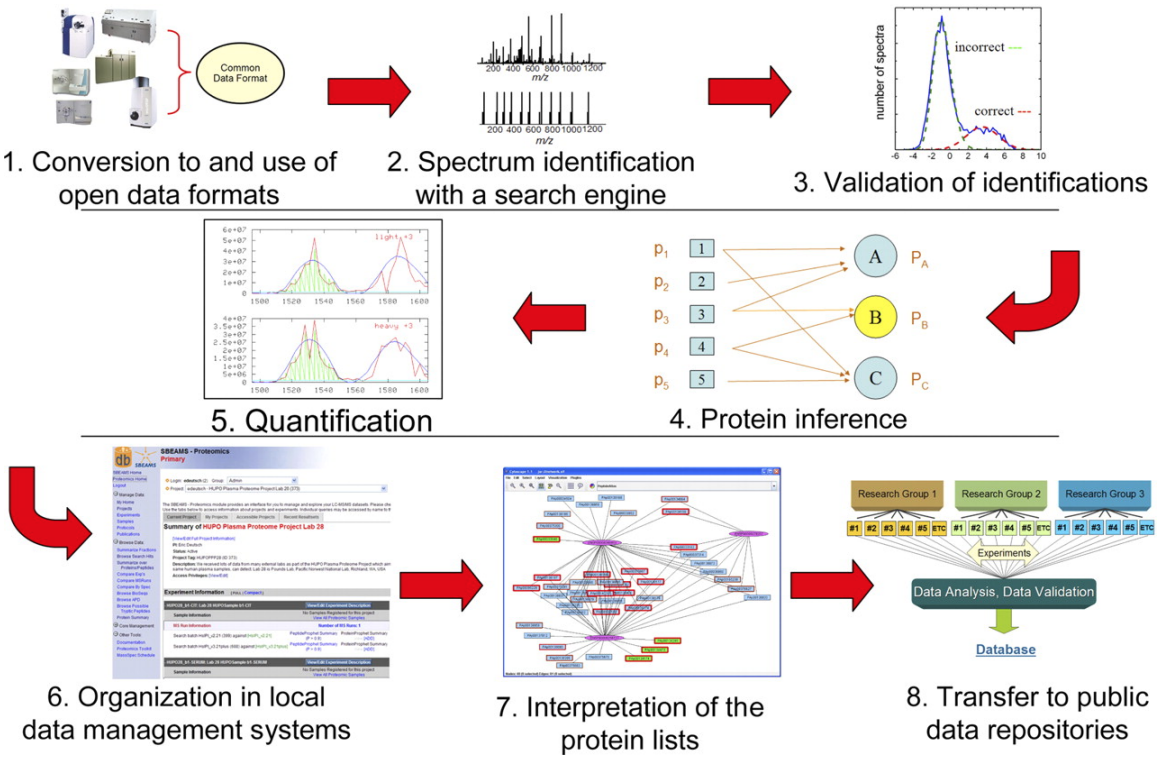


**Figure 1.2.8:** Tandem mass spectrometry workflow

a Laboratory Information Management System LIMS, or not). [20] includes a more complete description of this workflow and the most well-known software tools associated with each step.

A classification of the most frequent tasks based on the described workflow have been included in [84] in order to describe the contributions of different open source libraries in MS experiments. Below we describe the two software tools mentioned there that include several visualization techniques to aid in the different stages of the MS workflow. An extended list of visualization tools in MS/MS based proteomics can be found in section 5 of [48].

PRIDE Inspector

The PRoteomics IDEntifications database PRIDE, is a centralised repository where proteomics data is stored and shared. In order to submit data into PRIDE, a series of standards have to be followed, which ensures the high quality of the data. These types of resources have been used more often in part because some journals require that for an article to be published, its data should be stored in a public repository that follows community standards.

PRIDE Converter (`https://code.google.com/p/pride-converter/`) is a tool that provides support during

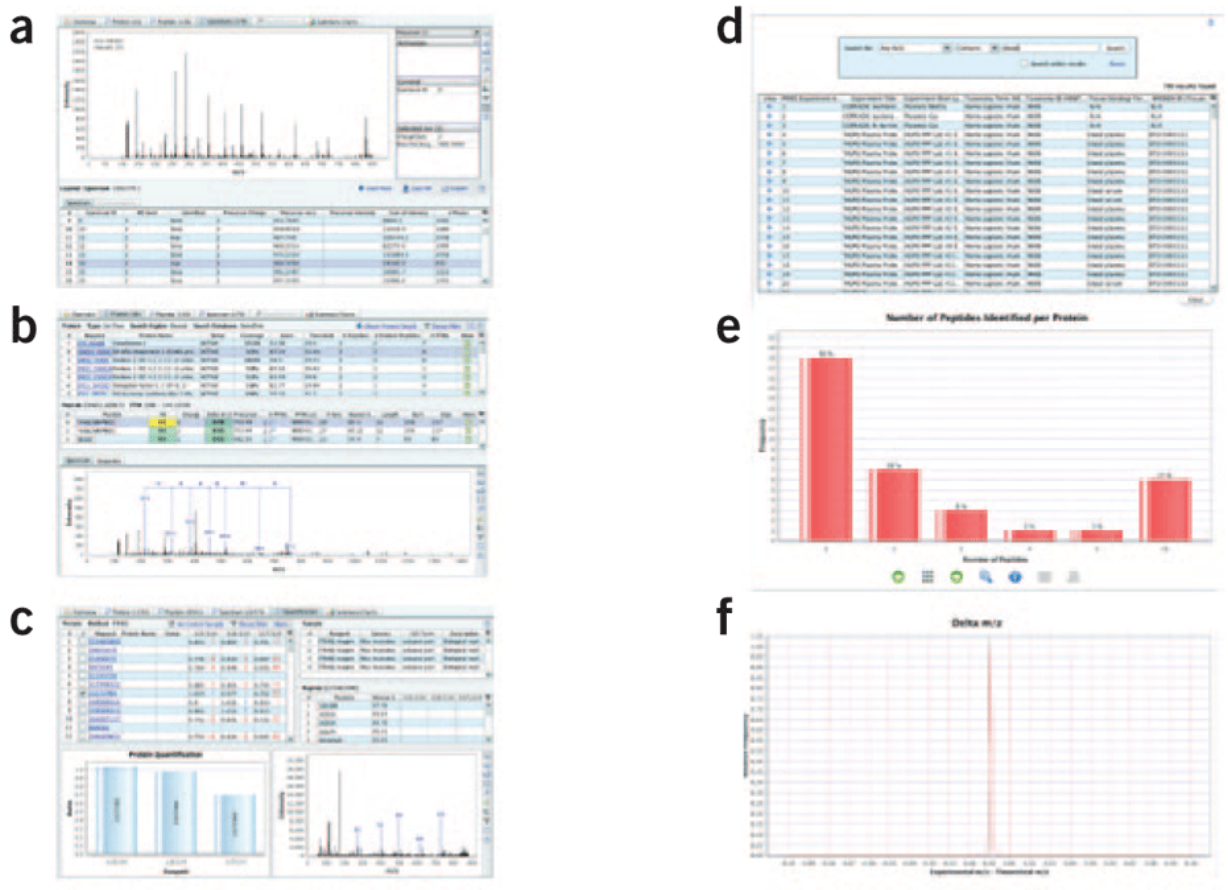


**Figure 1.2.9:** Snapshots of the PRIDE inspector toolset: (a) Section of the spectrum view tab. (b) Protein view tab, including the spectrum viewer showing MS/MS fragment ion annotations (only b ion annotations are shown). (c) Quantification view. (d) 'Search PRIDE' panel. (e) Number of peptides identified per protein chart. (f) 'Delta m/z' chart.

the process of submitting new data to PRIDE. The same team developed PRIDE Inspector following the success of the converter, and motivated by the fact that inspection and validation of reported results are of great importance during the review process [117].

Figure 1.2.9 is a group of some of the views of the PRIDE Inspector application. The user can load their own data or explore the public data from PRIDE using the search view(d). The proteins of the data set are listed (b) including their peptides, Post Transactional Modifications PTMs and corresponding spectra. The spectrum viewer (a) includes automatic annotations based on submitted fragmentations. Some aggregate charts (e) and (f) are also available to explore the information of each protein, and if the experiment includes quantification data, the view in (e) is enabled and the data can then be used to create comparative charts.

PRIDE Inspector was developed with modularity in mind, and as a consequence, libraries packaging

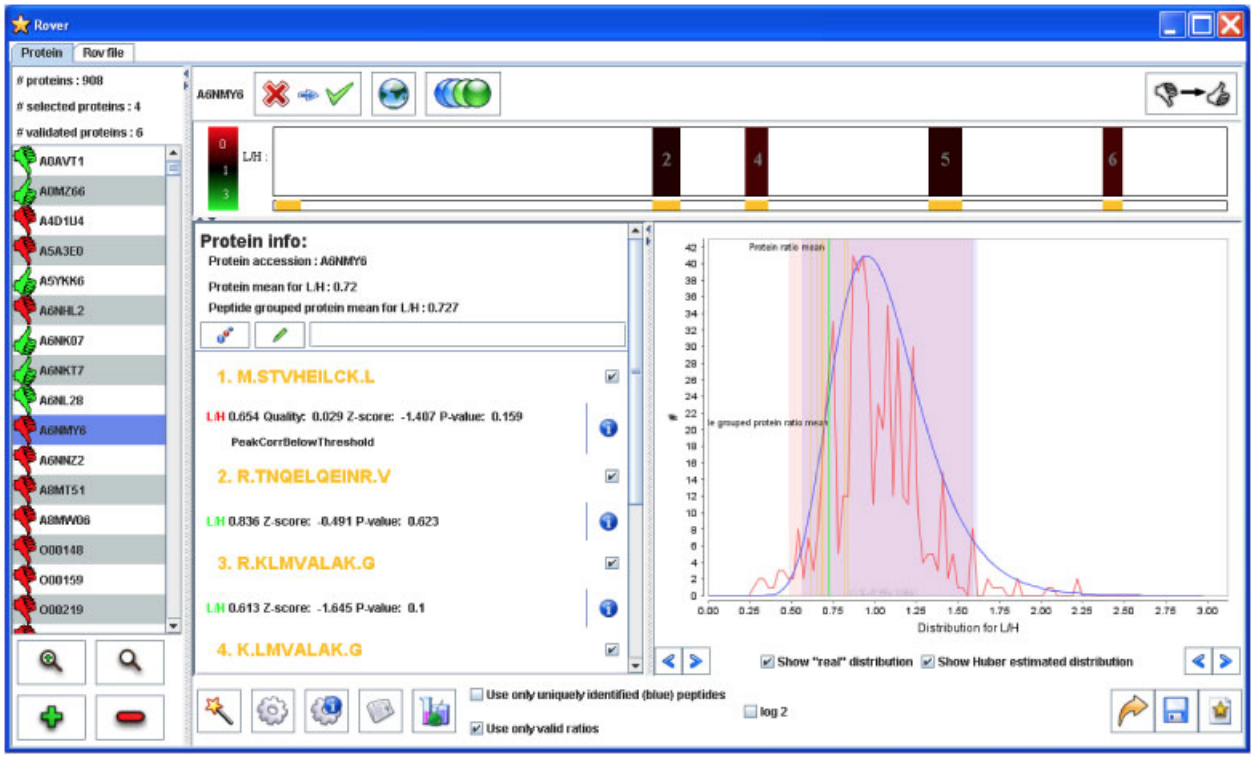


**Figure 1.2.10:** The main Rover interface.

some of the functionalities can be obtained independently. The visualization routines are accessible from a library called mzGraph that can be reused in other projects.

Rover

The open source Java application called Rover, focuses on quantitative proteomics data. It generates visualizations of this data that help the user in the process of selecting and validating algorithm-suggested regulated proteins in the context of an experiment [84].

Quantitive proteomics datasets are composed of two parts in Rover: peptide identifications and quantitative data. Rover includes a wizard-like menu that facilitates the input of these files. It also includes several parsers for some of the most frequently used formats of this type of data.

Once the data has been successfully loaded into Rover, a high level view of it is presented to the user so they can select and validate the regulated proteins.

Figure 1.2.10 is an snapshot of the main interface of Rover. It displays protein and peptide information as follows: on the left side there is a panel displaying all the proteins that are part of the experiments; glyphs are used to indicate if the protein is selected(thumbs up) or not(thumbs down), and it is colour

coded the protein is validated(green) or not validated(red). The top of the screen is the protein bar, where the peptides are drawn proportionally over a rectangle that represents the whole protein. The centre of the window is then divided again: the left side offers general information of the selected proteins and the right side displays the ratio distribution graph, where the selected protein and its ratios are shown in comparison to the reference set, using either the intrinsic distribution of the data or transforming the data to use the Huber distribution, which can give a new perspective to the analysed data [13].

All data generated with Rover can be exported for subsequent analysis.

*Microarrays*

Although Microarray techniques are considered by some to be out of the proteomics field because they deal with transcription data more often than with proteins themselves, this technology offers a high throughput solution to study the expression levels of genes on a particular sample. Although the level of expression of a gene is not enough to evaluate the number of proteins produced, it is considered a good indication.

There have been advances in microarray techniques that use proteins in order to complement MS studies [86]. Because of their high usage we will focus on visualization tools used in the analysis of DNA/RNA microarrays. In particular, we will describe two suites that serve to exemplify the most common scenarios where visualization techniques are applied to microarray data. This however is just a small sample of all the tools that are used to analyse this type of data.

Bioconductor

Bioconductor is a library for the programming language R, that aims to provide support for general research in computational biology. They have orientated their efforts towards the statistical analysis of microarray experiment's data, covering many of the usual tasks required: preprocessing, quality control, normalisation and downstream inference of biological and clinical questions [34].

The development of Bioconductor has been inspired by the achievements of other software that follow the principles of the Free Software Foundation. The authors consider that the adoption of these principles for the computational biology and bioinformatics fields is key to improving research in terms of transparency (i.e. exposure of the entire process), reproducibility and efficiency of development.

R is a programming language recognised by its numerical and statistical capabilities, which can be easily connected to many types of visualization graphs. R can be used for rapid prototyping and quick exploration of data. It supports multiple network and parallel computing protocols, as well as access to different database systems. These reasons attracted the Bioconductor community when they were choosing which language to use in order to implement their goals.

The Bioconductor project has invested much of its time into the creation of the infrastructure of the project, for example, by defining protocols for submission of new packages and their documentation; interfaces to the existing packages; documentation for both user and developer; exposure to the documentation from within the system; guidelines on how to use their source code repository; definition of good practices when developing a Bioconductor packages, etc.

Thanks to these efforts, Bioconductor now has a strong community of users with more than 3000 subscribers to their forum and over one million visitors to their website, on which is possible to find over 800 software packages (release 2.14). Their publication [34] has more than 6000 citations and many of the packages developed for Bioconductor have also been published in peer-reviewed journals [75].

In terms of visualization, the Bioconductor repository had 218 packages listed with the visualization tag by January 2015 (http://www.bioconductor.org/packages/release/BiocViews.html#___Visualization). Figure fig:bioconductor is a random selection of snapshots from several packages found on that list. This includes: (a) Aroma (http://www.aroma-project.org/screenshots/) provides a view similar to that of genome browsers in order to display copy-number amplification in a chromosome; (b) is a diagnostic plot generated by arrayQuality http://arrays.ucsf.edu/; (c) is a clustered heat map and box plot, which results from the analysis of cell abundance changes in subcommunities deployed by using flowCyBarhttp://www.ufz.de/index.php?de=32737; and (d) is part of the case study used to describe survcomp http://www.pmgenomics.ca/bhklab/software/survcomp a package created to compare the performance of survival/risk prediction models.

### Chipster

Chipster is a software suite that includes many of the most frequently used software tools for the analysis and visualization of microarray data and other high throughput experiments. It has a client-server architecture where the client is a graphical oriented interface that allows not only the display, but also interaction with visualizations. The server on the other hand is oriented to the execution and administration of the tools that a particular Chipster installation offers.

Cipster extensevely support the bioconductor package, allowing the editing of scripts to include functionalities from the R package. Other tools can also be imported to Chipster, however only the ones written in bioconductor can be edited through Chipster.

Chipster is an exploratory tool where different tools and visualizations can be used at any time, guided by the user's will. Moreover the execution history is saved as a workflow, and therefore it is easy to re-execute with different data or tune the different tool's parameters. The workflow files can also be saved and shared.

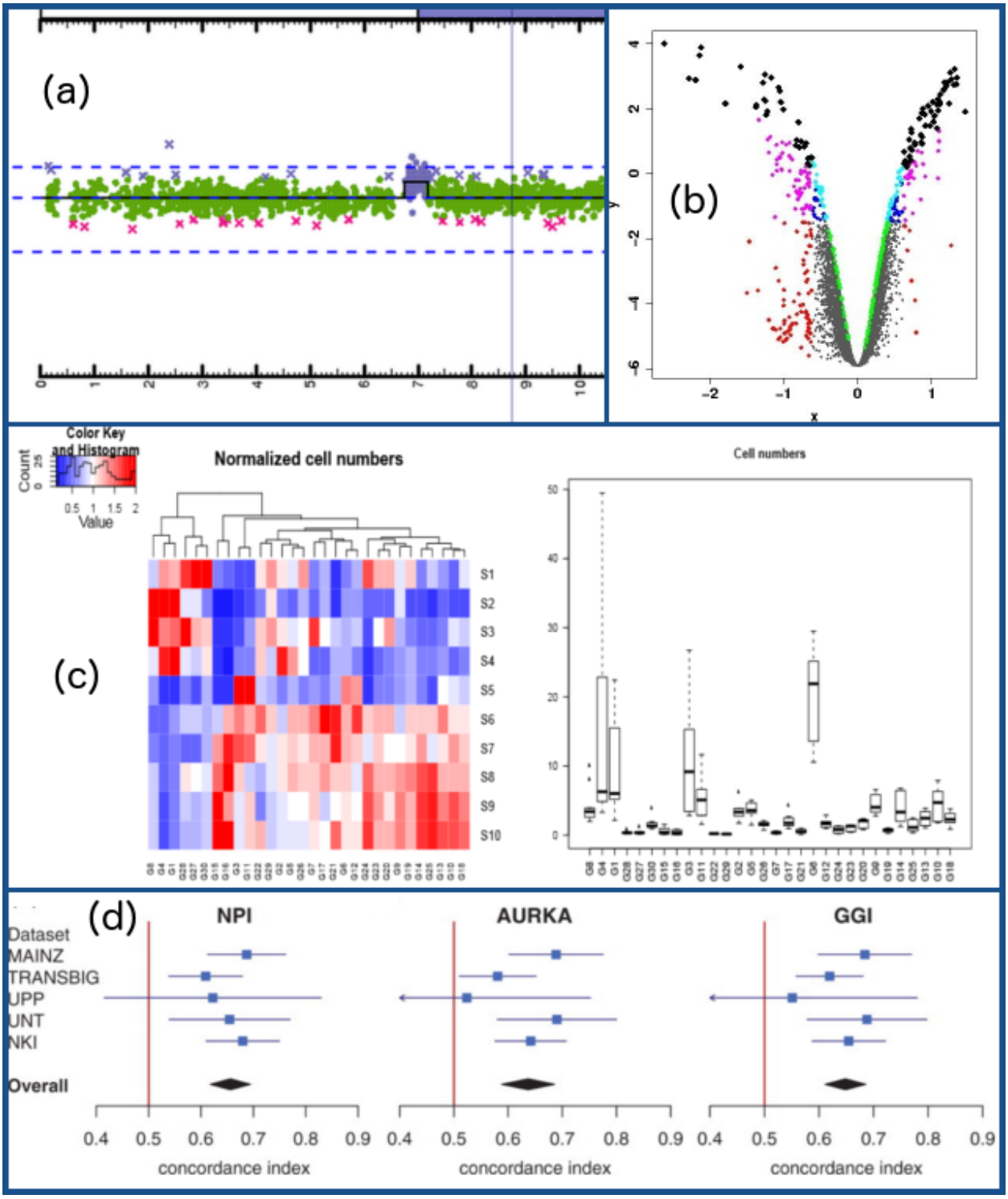


**Figure 1.2.11:** Visualizations created with Bioconductor's packages: (a) aroma. (b) arrayQuality. (c) flowCyBar. (d) survcomp.

There are more than 25 visualizations included in Chipster. The ones generated for the client software can be manipulated by the user, and they also serve as a way to interact with, select and filter the data. The visualization generated by Bioconductor and other imported tools are treated as static images and basic zooming and panning is supported. The interactive visualization include scatter plots, volcano plots, venn diagrams, heat maps, self organised maps, among others; and can be used in different stages of a microarray experiment, e.g. normalisation, quality control, filtering, statistical testing, clustering, annotation, etc. [53].

*Protein Annotations*

There are several projects that provide a visualisation similar to the one offered by genome browsers, but in which reference and features are protein data. The challenges for these types of applications can be slightly different to their genome browser analogues. For example one the biggest challenges of the latter technology is how to deal with the length of a genome, and therefore those applications have developed strategies for smart panning navigation, only loading by-request and offering several graphic interface gestures to navigate through a chromosome as seamlessly as possible. For protein browsers the length is not a major concern, as proteins are usually less than a thousand amino-acids long, and at present most computer screens use resolutions higher than 1024x768, so the inclusion of a representation of the whole protein on the screen should be relatively simple.

On the other hand, the number of sources of protein data can be quite large, and therefore some strategies are required to deal with the vertical growth of the visualisation. For example, grouping annotations on the same track, or some kind of vertical panning/scrolling. Below we briefly describe some of the most recognised protein annotation browsers, and in section 2.3.1 we will discus Dasty3, a protein feature viewer in which we have contributed during this PhD.

PFAM

The Protein Family Database (PFAM) project's main goal is to provide information for protein families and domains. The database is divided in two subsets: PFAM-A contains curated information and PFAM-B is an automatically generated database.

The PFAM website provides several ways to access this data, one of them being the feature viewer, where besides PFAM own annotations, it allows users to get information from other sources and to put it in the same context. Figure 1.2.12 shows the result of a query in the PFAM feature client. The first track contains all the annotations about families and domains that the PFAM database has. The next set of tracks are the rest of the annotation sources that PFAM provides. Finally all the features of external data

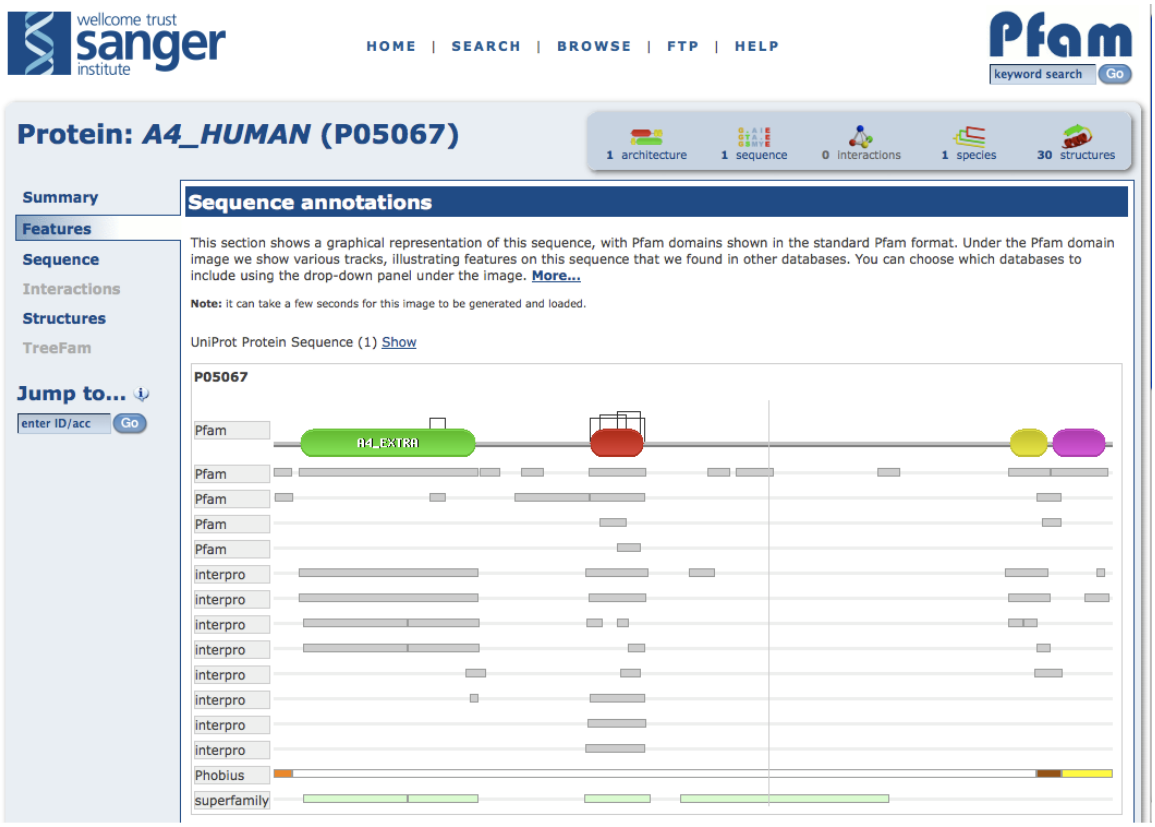


**Figure 1.2.12:** *PFAM*: Web widget, which places special emphasis on the visualization of the annotations of protein domains

sources that the user has selected for that query are shown [27].

InterPro

The InterPro database is a public resource used to classify sequences into protein families which can be annotated, and subsequently used to infer features of non-annotated proteins. InterPro combines the data from different datasets, and has grown from four sources in 1999 to eleven in 2014. The combined resource provides information about protein families, domains and functional sites.

The central concept of InterPro are protein signatures, which are representations of groups of protein sequences based on similarity. These are annotated with a name, a descriptive abstract and gene ontology (GO) terms. Functional features of a signature can be used to describe a protein by identifying the existing signature in the target protein. This approach has predicted the function of almost 50 million individual proteins [74]

Figure 1.2.13 is a snapshot from the InterPro website, displaying 16 individual signatures from seven

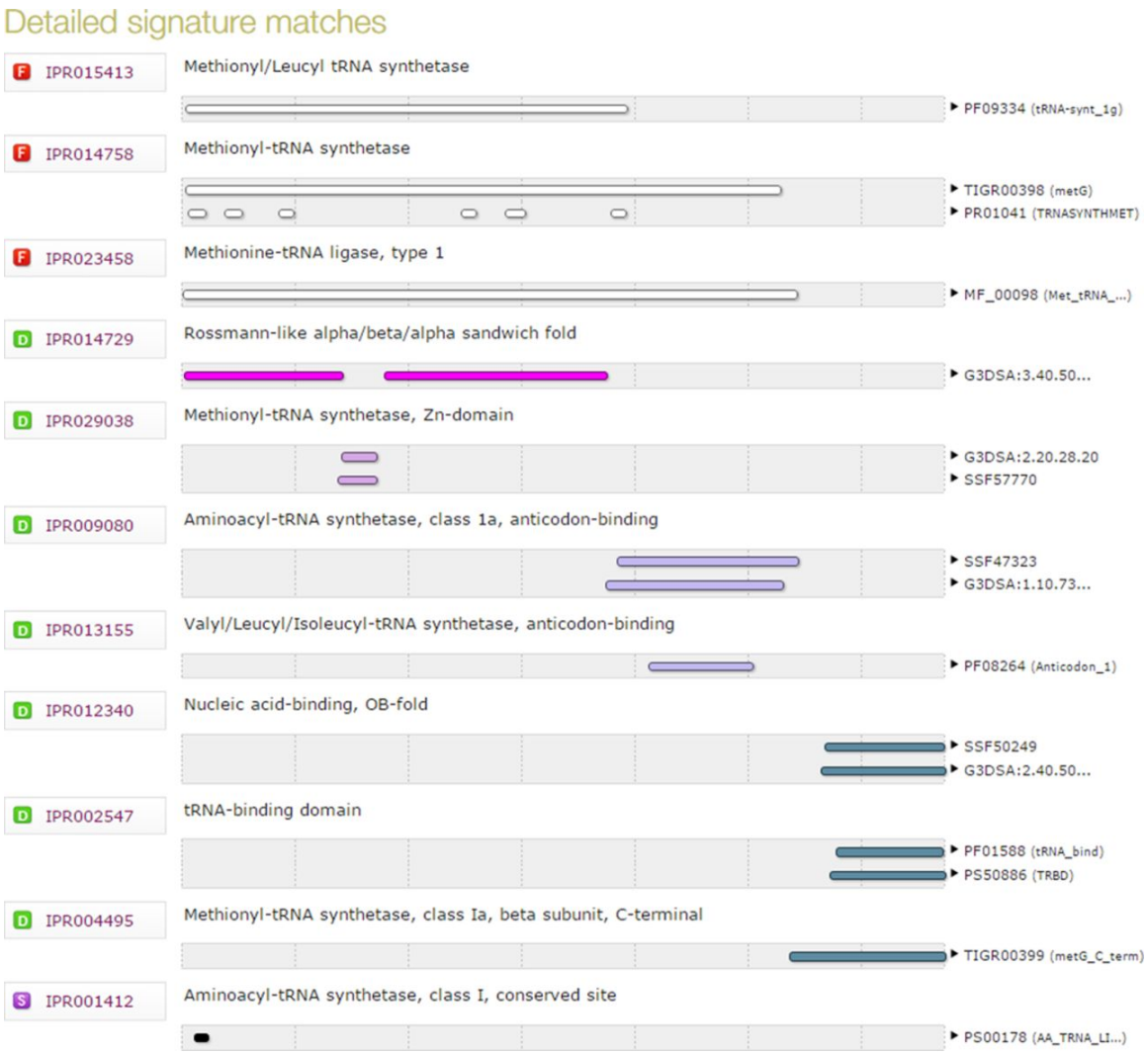


**Figure 1.2.13:** Detailed InterPro member database match data for UniProtKB entry q3JCG5

sources, that have been grouped into 11 entries. It uses a similar visual approach to other protein annotation viewers, in which features are located proportionally to the reference, but in this case the reference is not another track on top, but instead is a box where the annotations are included. In this way the reference is always visible, no matter how many entries are on display or how far down have the user has scrolled. However, this view takes more space and as a consequence, less tracks can be displayed simultaneously.

### 1.2.3 Protein Interactions

Interaction between proteins is one of the most important mechanisms in the execution of cellular functions. The study of these interactions has provided insight into the functioning of an organism's

processes.

The number of reported Protein-Protein Interaction (PPI) networks has grown considerably, partly due to advances in high-throughput experimentation and partly due to the new predictions that result from these empirical data. For example, as of January 2015, *Homo sapiens* had over 300000 Protein-Protein interactions (PPI) registered in the Interologous Interaction Database, which is only one of the many public resources where protein interactions can be accessed for different species.

Data repositories for PPI data can be classified into three groups: (i) Primary interaction databases, where curated experimental results are deposited, for example: IntAct [57], MINT [66], BioGRID [109], DIP [98] and HIPPIE [99]; (ii) computational predicted databases, for example, DIMA [69], PIPs [70], PrePPI [124] and PRISM [79]; and (iii) databases integrating both types of data, for instance, GeneMANIA [119], FunCoup [101], I2D [77] and STRING [111].

The volume of data that PPI repositories have reached has made their analysis and understanding a challenge that can be enriched by the use of visualization techniques. PPI networks have been found to follow a behaviour that in graph theory is referred to as *scale-free*, which uses a few highly connected nodes and many nodes with few interactors. This feature allows the user to predefine visualization strategies to highlight certain characteristics.

The list below was identified in [2] as the main requirements for the visualisation of protein interaction networks:

- Clear rendering of network structure and sub-structures, such as dense regions or linear chains;
- Fast rendering of huge networks;
- Easy network querying through focus and zoom;
- Compatibility with the heterogeneous data formats used for PIN representation;
- Interoperability with PPI databases, allowing the automatic querying of single or multiple databases using existing middlewares;
- Integration of heterogeneous data sources, e.g. functional information about proteins extracted from biological ontologies.

There are several tools that allow the exploration of PPI data, providing different methods to visualize protein network interactions. [2] evaluates some of them by considering the requirements above. Other surveys can be found in the literature where comparisons and descriptions of several PPI tools are included [32, 83, 110]. Below, we describe some of these and in section 3.2 we explore in detail PINV, a web based PPI visualization tool.

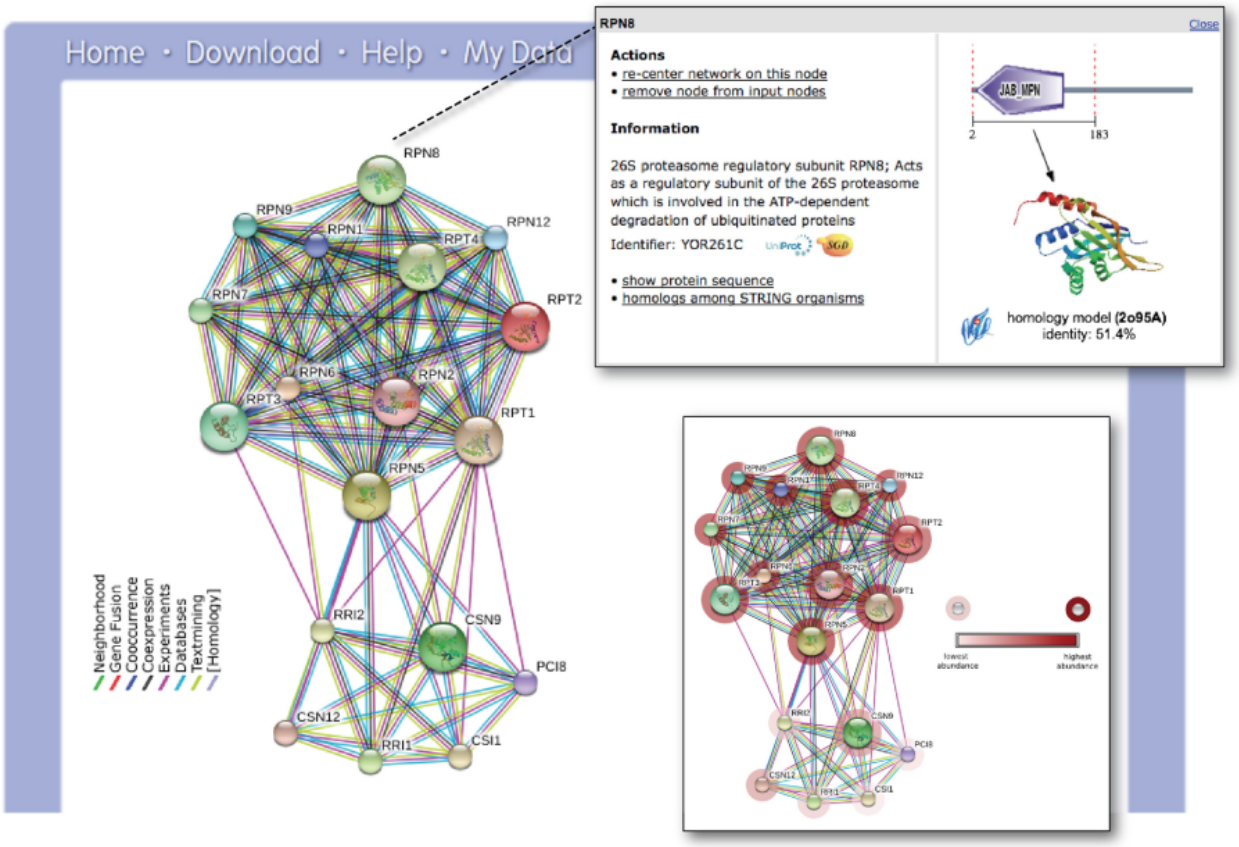


**Figure 1.2.14:** Combined screenshots from the STRING website

*STRING*

STRING is a repository for PPI data that includes both primary and computationally predicted interactions. It differentiates itself from other resources such as those mentioned above in three ways: coverage (more than 2000 organisms), the inclusion of text mining data and the metadata linked to the interactions (e.g. protein domains and structures) [29].

STRING includes a viewer developed in Adobe Flash that allows the direct manipulation of the graph based on mouse interaction. The network to be displayed in the viewer is the result of a user query of either one or multiple proteins. However it can also be invoked by a call to one of the resources that are cross-linked to STRING.

Each protein node includes a preview of the 3D structure (if available). When the user clicks on the node, a popup window opens with extra information about that protein, including links to external resources such as PDB or UniProt. The interactions between the proteins are represented by multiple lines representing the different sources of evidence; clicking on an interaction opens another popup

window with more details on the evidence data. See figure 1.2.14 for an example of the viewer including a pop-up window for protein details.

One of the most recent developments in STRING is the creation of a bioconductor package that can access STRING information directly from the R programming language in order to allow automation of the data collection from this resource. This feature is an addition to previously released data access services: REST and the availability to download version based compressed files of all the interactions in STRING [111].

*Cytoscape*

Cytoscape is "*a free software package for visualizing, modeling and analyzing molecular and genetic interaction networks*" [11]. It is arguably the most used tool to create visualisations of biological networks, and has a general purpose architecture that provides the environment to model biomolecular interaction networks. We consider that besides being a useful software that reaches the expectations of a network visualiser, there are two major factors for its success: Cytoscape has a well designed architecture with robust support for plugins (called apps since Cytoscape 3.0) and it follows an open source philosophy.

The success of the project is apparent in its growth: In the welcome letter (`http://cytoscape.org/cy3_welcome_letter_v12.pdf`) of the project to their users dated on November 7, 2014 they report to have an average of 12000 downloads of the application per month and counted that it has been opened 2000 times per day worldwide. In addition, Cytoscape now has over 200 plugins available in its App Store, a clear example of its collaborative environment, where scientists/developers can contribute to the network analysis toolset by adding new plugins.

The core of Cytoscape is their network graph, which uses the previously mentioned node-link technique, where the nodes can be any type of biological entity (e.g. genes, proteins, organisms) and the links are the pair-wise interactions [103]. It is important to note that the decision to use a generic representation for nodes and links is a factor that has attracted users from the different domains of the biological sciences. This is as opposed to tying the generic representation to protein or genes for example as has been implemented by similar projects.

The network graph can be manipulated by the user selecting where to locate a node, however this task needs to be automated in order to save time and be able to deal with many nodes. Cytoscape provides many automatic layouts on its basic installation and there are more options available as plugins.

The network can be enriched by two means: attributes and annotation. Attributes are name-value pairs that can be associated with either a node or a link. The concept of an annotation in Cytoscape is similar to that of the attribute, because it is also an association to a network entity, but annotations require the use of ontologies (e.g. gene ontology) in order to follow a hierarchical structure for filtering

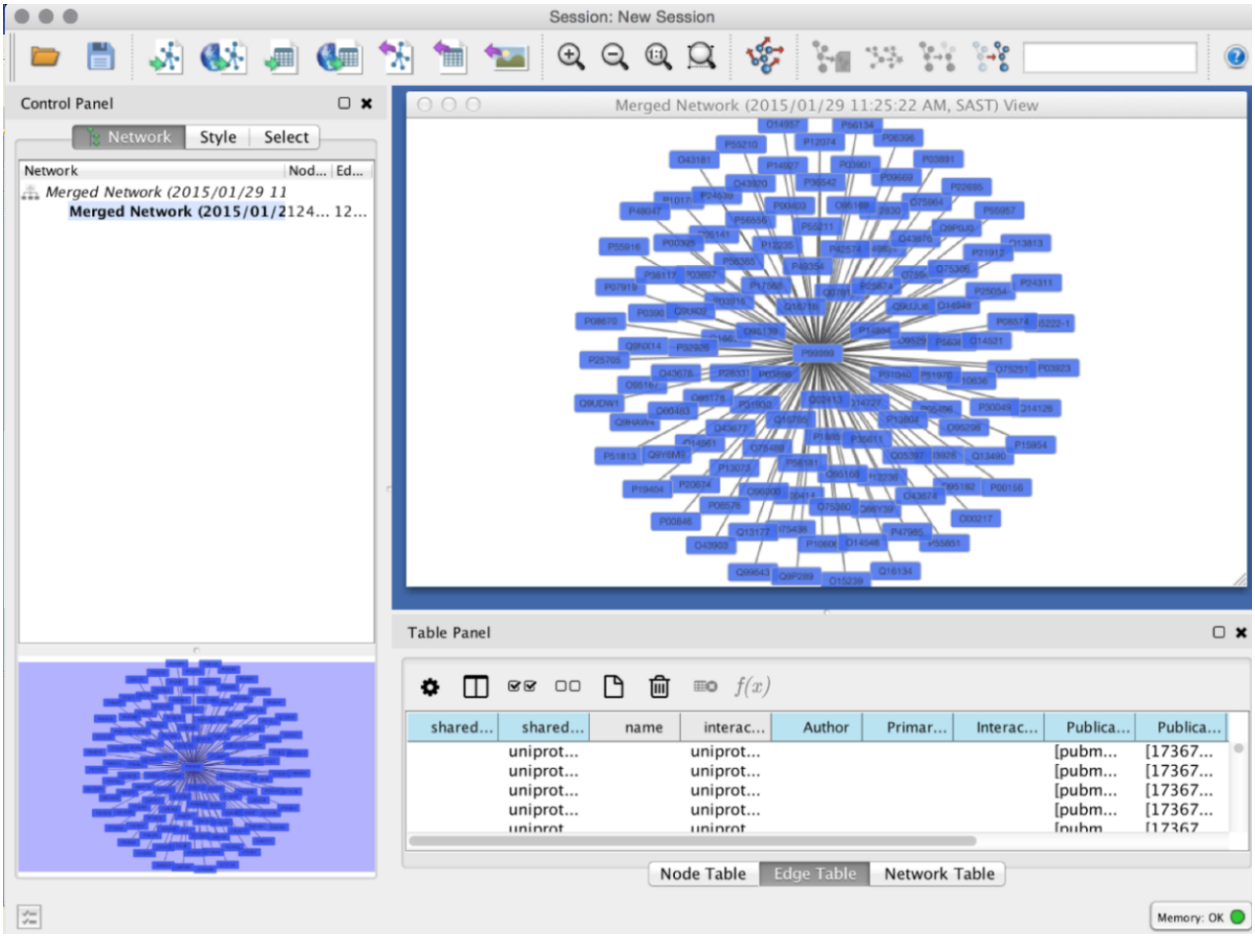


**Figure 1.2.15:** The Cytoscape Desktop.

and manipulation. This information can be added by different means: (i) manually included by the user (i.e. by using the spreadsheet interface), (ii) calculated by the tool or one of its plugins (e.g. network attributes such as the degree of a node), (iii) extracted from an on-line resource (e.g. getting the GO terms from UniProt), (iv) imported from a local file (e.g. expression values per protein) [92].

Figure 1.2.15 shows the graphic interface of Cityscape 3.2.0 running on a Mac OS X. Quick access to the most frequently used tools have been grouped in the top bar. The left panel provides a hierarchical structure to navigate though the current networks and includes a zoomed out view of the graph; the main panel is the Cytoscape canvas where the network graph is on display, and at the bottom-right is the spreadsheet interface that allows to browse the attributes.

The most basic uses for attributes and annotations are to filter the network by selecting only the nodes with an attribute of interest. They can also be used to map the information onto the graphic by altering colour, shape and size of the nodes and links, according to the value of an specific attribute, allowing the

user to visualise several types of data within a single view. However the use of this metadata can also be extensively used by the plugin's developers in many other different and complex scenarios.

Besides the growth of the application due to the development of new plugins, the core application is continually updated by adding new features and fixing bugs. For example, two major features were included when version 2.8 of Cytoscape was released: the possibility of using an image to represent the node, and the support for equations on the spreadsheet view of the network. Although both functionalities were previously available via Cytoscape's API, their inclusion on the main application facilitates their use by non-programmers [106].

A protocol describing a common workflow in Cytoscape was published in [11]. In it, a classification of the most common tasks is included and used to structure the document. The five tasks identified were:

- *Obtain network data*: The user has several options to do this, including loading local files in one of the multiple supported formats, importing data from an on-line resource such as MINT or STRING or invoking text-mining plugins that analyse literature resources such as PubMED to infer reported interactions of genes or proteins of interest.
- *Explore network and generate layout*: As mentioned before, several automatic network layouts are included in Cytoscape.
- *Annotate with attribute and expression data*: The results of an expression experiment can be mapped to an existing network by, for instance, colouring the nodes in a scale from green to red, in order to easily identify over and under expressed genes in the graphic.
- *Analyse network features*: The entities included in the network can have associated functional classes, and the discovery of clusters or hubs can be relevant to the biological understanding of the system.
- *Detect enriched functions*: The same information can also be used to infer functions on other entities that were not well annotated.

Given the growing number of apps, the Cytoscape team developed the App Store, which was released together with the version 3.0 of Cytoscape in 2013. This service attempts to provide a showcase for developers to display their contributions to the Cytoscape community, and for users to find the right tool to complement their research. App authors can include screenshots, descriptions, links and other metadata in order to highlight the features of the app and facilitate its use. App store users can navigate though the categories, search by their attributes, or select one from the list of featured apps. Besides providing information, the App Store also allows apps to be installed and upgraded with a single click [68].

*Web based PPI tools*

As stated in [32] "*there is a trend toward web-based applications, often coupled tightly to underlying databases*". STRING (described above) is a clear example of this tendency, where an effort to integrate PPI information in a centralised resource is combined with a tool that allows the search, exploration and visualisation of such resources. Most interactive online visualization tools follow this path, only allowing the view of networks that are preloaded, meaning that the user cannot view a network he/she has generated.

The online version of Graphle [47] is limited to graphs generated by bioPIXIE in yeast, MEFIT in *E. coli*, or HEFalMp for human data. STITCH [61], a project for chemical-protein interactions, integrates interaction information from various sources in addition to in-house predictions. Both STRING and STITCH use the same library to display their data. However, the network visualized in both databases is barely customizable. The public implementation of VisANT [44], a workbench for the integrative analysis of biological networks, is based on the Predictome database. Although these tools are available on the web, they require third party software (e.g. Adobe Flash, Java Virtual Machine) and therefore their accessibility is more limited than native web applications (i.e. developed using recent web standards).

Another alternative is Cytoscape Web [67]. Its introduction tutorial (`http://cytoscapeweb.cytoscape.org/tutorial`) requires users to code in JavaScript indicating that this tool is mainly intended for developers to display networks on the web, and not for the scientist who has a network to visualize. Cytoscape Web also uses Adobe Flash for the generation of the graphics. A more recent development is Cytoscape.js (`http://cytoscape.github.io/cytoscape.js/`), which uses the HTML5 component called canvas, and therefore its only dependency is a modern web browser. The principles behind this project is to offer a modern web toolset to display interactions. Similarly to its predecessor, Cytoscape.js is a library for programmers, and is intended to be a complement to the Cytoscape Desktop.

A web native application that supports the visualisation of networks and is intended for non-programmer users is Pclust. Its objective however is not the visualisation of PPI networks. Pclust is a tool that assists in the inference of functions of new proteins from their homologs, using a technique known as all-against-all pairwise similarity. The network is built by looking for similar functionality between proteins. The set of proteins can be either entered by the user or detected by executing a BLAST (Basic Local Alignment Search Tool) against NR. The resulting network is then enriched with related literature through the Seq2Red server (`http://prodata.swmed.edu/seq2ref/`). The visualised network creates clusters of proteins with similar functionalities. Referenced proteins are highlighted, which helps to solve ambiguities in the deduced functions [65].

### 1.2.4 Discussion

The projects described above gave us a clear insight into the influence that visualisation techniques have on molecular biology research. We have only mentioned a few cases in the field of genomics, proteomics and protein-protein interactions networks, but the visualization techniques have been applied to many other fields such as metabolic pathways (and system biology in general), population genetics, metagenomics, etc.

It is important to note that each scenario required a different visualisation technique, from simple two dimensional plots used to display complex statistics (e.g PCA), to highly detailed tools that combine more than one visualisation technique, for example, genome browsers.

Genome browsers have evolved from simple displays of genes and transcripts to become visual integration tools of biological data. Tracks can be preprocessed using many combinations of statistics and data analysis, and its visual representation can vary from simple boxes to histograms, heat maps, and many other charts. This is not unique to genome browsers and the exploration of biological data have required the use of the whole range of visualisation strategies in an attempt to elucidate specific biological cases, but ultimately to contribute in the understanding of life.

One of the most appropriate ways to evaluate the success of software applications is through the number of users, developers and collaborators involved in the project. A large community can detect errors and improvements faster, and therefore act quicker, which ultimately makes the software improve faster, this in turn means, more users are attracted to build an even bigger community, and to restart this positive cycle.

A common denominator in successful visualisation software is that the tools are adaptable to the needs of each research project. For example, genome and protein browsers allow the inclusion of third party data in context with their original sources: Bioconductor promotes the development of new packages by offering a robust infrastructure and Cytoscape is highly extendable by the means of plugins and its App store.

The publication of scientific articles has become one of the major motivators for researchers, because papers imply recognition of their efforts, and they are highly valued in the assignation of grants and job positions. We considered that this is another factor that can play a part in the success of software in bioinformatics. A good software should offer a robust toolset that can be improved/extended, by, for example, using the plugin philosophy. These improvements can be guided by scientific needs which are publishable on their own. The publications give visibility to the software, attracting more users to restart the positive development cycle.

Take for example, Cytoscape: this software includes a reasonably large set of features for the analysis and visualisation of biological networks. Thanks to its plugin architecture it attracts scientist that are

interested in doing something beyond the original scope of the tool: they can implement their ideas by reusing the features offered by the software, avoiding wasting time reinventing the wheel. Moreover, their contribution can be novel and interesting enough that it can be published in a peer reviewed journal. Ultimately, Cytoscape gets a new feature to offer and more visibility thanks to the publication.

This approach gives the community constant improvements in their research tools, that are complying with the high standards of peer-reviewed publications.

From the projects mentioned in this chapter, Bioconductor, Galaxy and Taverna have implemented similar strategies with a number of articles published around their technologies, and growing communities supporting their software.

Lastly, it is important to mention that in this chapter we are using a completely artificial separation between integration and visualisation, because it is hard to find projects that are exclusively on one side or the other: central repositories usually have visualisation methods to explore their data (e.g. STRING, PFAM, InterPro) and many visualisation tools are about integrating data (e.g. Genome Browser). This PhD project has focused on this area, and our contributions are mainly focused on either integration or visualisation, and have been classified as such for the following two chapters. Nonetheless, the overlapping nature of the bioinformatics needs between these fields is clear and therefore the feature set of the tools described in the following chapters reflects this.

*La verdad no penetra en un entendimiento rebelde. Si todos los lugares de la tierra están en el Aleph, ahí estarán todas las luminarias, todas las lámparas, todos los veneros de luz.*

Jorge Luis Borges, El Aleph

# 2

# Integration of Information in Bioinformatics

**First author publications** :

1. Gustavo A. Salazar, Leyla J. García, Philip Jones, Rafael C. Jimenez, Antony F. Quinn, Andrew M. Jenkinson, Nicola Mulder, Maria Martin, Sarah Hunter, and Henning Hermjakob. Mydas, an extensible java das server. *PLoS ONE*, 2012
2. Gustavo Salazar, Rafael Jimenez, Alexander Garcia, Henning Hermjakob, Nicola Mulder, and Edwin Blake. Das writeback: A collaborative annotation system. *BMC Bioinformatics*, 12 (1):143, 2011. ISSN 1471-2105. doi: 10.1186/1471-2105-12-143. URL http://www.biomedcentral.com/1471-2105/12/143
3. Gustavo Salazar, Nicola Mulder, and Edwin Blake. DAS Writeback: A Collaborative Annotation System. Master's thesis, University of Cape Town, 2010. URL http://pubs.cs.uct.ac.za/archive/00000609/

**Coauthor publications** :

4. Jose M Villaveces, Rafael C Jimenez, Leyla J Garcia, Gustavo A Salazar, Bernat Gel, Nicola Mulder, Maria Martin, Alexander Garcia, and Henning Hermjakob. Dasty3, a web framework for das. *Bioinformatics*, 27(18):2616–2617, 2011

**Author's Contibutions** :

- *1*: Conceived and designed the experiments: GS. Performed the experiments: GS AJ. Wrote the paper: GA LG PJ RJ. Critical revision of the manuscript for important intellectual input: RJ AQ AJ NM MM SH HH. Technical and material support: AJ NM MM SH HH. Supervision: NM MM SH HH. Study concept: GS LG PJ AQ RJ HH. Architectural design: PJ GS. Software development: GS LG PJ AQ. Evaluation of the compatibility with DAS protocol: AJ.
- *2*: Critical revision of the manuscript for important intellectual input: RJ, AG, HH, NM and EB. Technical and material support: HH, NM and EB. Study supervision: HH, NM and EB. Study concept: GS, RJ and AG. Architectural design: GS and EB. Software development: GS. Drafting of the manuscript: GS. Design of the usability experiment: GS, NM and EB. All authors read and approved the final manuscript.
- *3*: MSc theses by GS, Supervision by EB, Co-supervision by NM
- *4*: Critical revision of the manuscript for important intellectual input: JV, RJ LG,GS, BG, NM, MM, AG and HH. Technical and material support: HH, NM, AG and MM. Study supervision: HH, NM, AG and MM. Study concept: RJ and HH. Architectural design: JV, GS, BG and RJ. Software development: JV. Drafting of the manuscript: JV, AG and LG. All authors read and approved the final manuscript.

*The data analysed in bioinformatics comes from diverse and heterogeneous sources*, for example, the data might be captured from wet-lab experiments or deduced from *in-silico* procedures. It can refer to nucleic acid information and sequenced data, but also to expression levels and other protein related information. It is possible to analyse isolated organisms or to gather information from multiple species. It all depends on the purpose of the research and the availability of data, however it is almost inevitable to have to integrate more than one of these sources in order to tackle today's research challenges.

This chapter is focused on the contributions made during this PhD to the integration of data using the Distributed Annotation System. The first section describes MyDas, a server tool that facilitates the publishing of DAS sources. A proposal to extend DAS in order to support collaborative annotation is described in section 2.2. We have grouped our participation in several client-side projects with DAS into section 2.3. Lastly we discuss the impact of these projects, together with the present and future status of DAS.

## 2.1 MyDas

The research in this section has been published in the paper [95], referenced as 1, at the beginning of this chapter. Authors of this paper are Gustavo A. Salazar, Leyla J. García and Philip Jones (first authors). The co-authors provided input in line with their roles as supervisors. Parts of this section are based on the work of collaborators and this is indicated clearly below.

The contributions to the software development process of MyDas are as follows: the first version of the project was developed by Philip Jones with the collaboration of Anthony Quinn for the XSLT component. A second development cycle together with the support of the DAS version 1.6 was executed by Gustavo A. Salazar. Current maintenance of the software is led by Leyla J. García. As an open source project, it has received contributions from other developers, but the authors mentioned here have been the leaders of the project during its various stages.

### 2.1.1 Overview

As of January 2015, there were over 1500 sources registered in the DAS registry, and although not all of them implement the same capabilities, they all follow the DAS protocol with almost half of them updated to the latest version DAS 1.6E. There are common tasks among DAS sources: parsing, capture of arguments, exception handling, XML creation, dealing with the HTTP protocol, and more. Therefore, software specialised in these tasks is needed in order to allow data providers to focus on their specific cases.

MyDas is a software tool that assists in the process of publishing biological data through the DAS

protocol, described in 1.1.2. Data providers are required to implement an adaptor that connects the logic of MyDas with the data itself, and from there MyDas executes all the HTTP interfaces, XML encodings and other required operations to support all the DAS capabilities.

In the following sections we describe the architecture of MyDas, and present some examples of existing DAS services built upon MyDas.

### 2.1.2 Design and Implementation

MyDas is a Java Servlet Application which accepts HTTP requests, typically with one request for each command in the DAS specification. Responses are valid XML documents. MyDas is a Java 1.6 application, and it runs on a Java servlet container such as Tomcat (`http://tomcat.apache.org/`) or Jetty (`http://jetty.codehaus.org/jetty/`).

Figure 2.1.1 illustrates the architecture of MyDas, which is clearly divided between core and external control. The MyDas core is in charge of all the common tasks and the external control is what needs to be input by the data provider, for instance its storage system (e.g. a relational database or flat file), or the strategy used to query the data (e.g. in-memory or using pre-indexing), etc.

The development process of MyDas is supported by several tools, for example, it uses Maven (`http://maven.apache.org/`) for automatising the software built, control of dependencies, test and deployment. It also uses JUnit (`http://www.junit.org`) to create a set of unit tests to ensure the quality of the software. Software workers are implemented on the repository machine to execute the tests when new commits are submitted which notifies the main developers in case of failure.

The information required to use the external control components should be described in the configuration file, including data such as the URI, title and the relative path to the data source adapter. Then, the configuration manager makes the user options available to both the MyDas core and the data source implementation.

The elements of the DAS specification have been mapped into a Java object model, which must be used by the data source developer when creating an adapter. In this way , the core of MyDas can use the same subroutines to deploy heterogeneous data into DAS.

In order to facilitate the implementation of data sources, a template project is available with examples of both reference and annotation servers:
`https://code.google.com/p/mydas/downloads/detail?name=MyDasTemplate-1.6.7.zip`.

### 2.1.3 Instances

MyDas is being adopted by different data providers, including UniProt, InterPro and PRIDE.

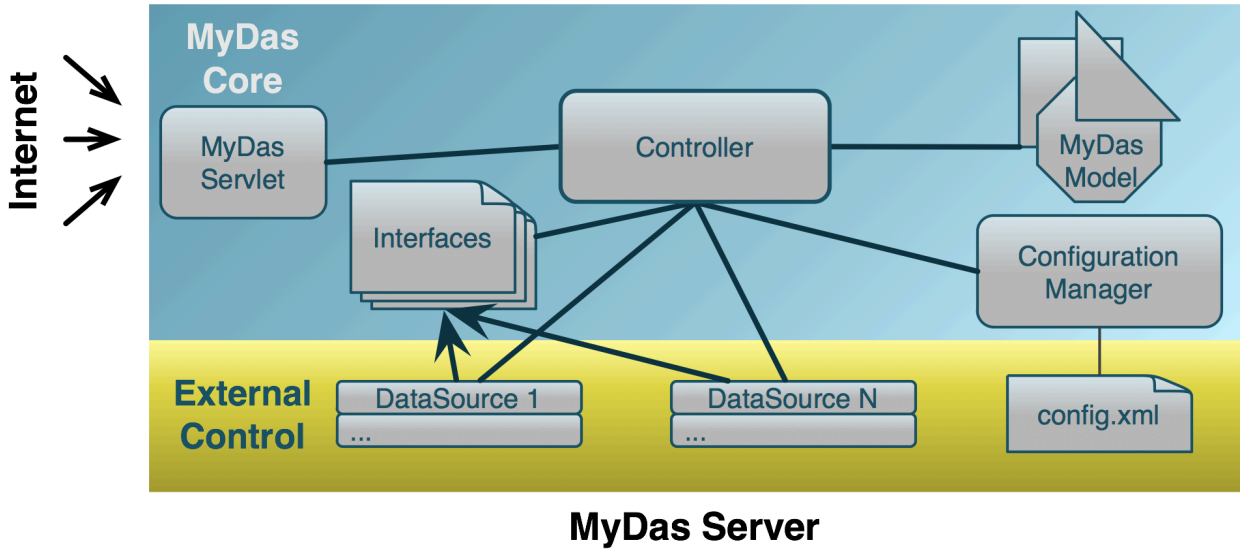


**Figure 2.1.1:** MyDas Architecture. The requesters can interact with MyDas through the servlet, which communicates the commands to the Controller. The Controller knows which Data Sources have been implemented by querying the Configuration Manager. Data Sources should implement at least one of the provided Interfaces. MyDas internally implements the DAS model.

UniProt (Universal Protein Resource)[15] is a comprehensive catalogue of protein sequences and functional information. It consists of different databases, each optimized for different uses. The UniProt Knowledgebase (UniProtKB) is an expertly curated database providing a central access point for integrated protein sequence information. The UniProt Archive (UniParc) is a non-redundant sequence repository of all publicly available protein sequences. UniProt DAS (`http://www.ebi.ac.uk/das-srv/uniprot/das/uniprot`, `http://www.ebi.ac.uk/das-srv/uniprot/das/uniparc`) acts as a reference and annotation server, providing access to up-to-date information and allowing queries by UniProtKB and UniParc accessions numbers. There are currently more than 50 Data Sources that use UniProt DAS as a reference.

The InterPro database of predictive protein signatures is used for the classification and automatic annotation of proteins and genomes [46]. InterPro provides several DAS data sources: DS_327 (`http://www.ebi.ac.uk/das-srv/interpro/das/InterPro` ) serves matches that have been calculated to the predictive models supplied by the InterPro member databases for all UniProtKB protein sequences. DS_1028 (`http://www.ebi.ac.uk/das-srv/interpro/das/InterPro-matches-overview`) serves these matches resolved to the InterPro entries that integrate the member database signatures (providing a compact summary view of the domains, families and sites predicted for each UniProtKB sequence). Finally DS_1029 (`http://www.ebi.ac.uk/das-srv/interpro/das/InterPro-UniParc-matches`) serves matches to member database signatures that have been calculated for UniParc protein sequences.

PRIDE DAS 1.6 (http://www.ebi.ac.uk/pride-das/das/) provides protein and peptide identifications together with supporting mass spectrometry evidence [116]. The information from PRIDE has already been shared using BioMart[59], therefore the strategy used to make it public to the DAS community was to develop an adaptor using MyDas to take the information from this source.

### 2.1.4 Tutorials

We have developed three tutorials on how to use MyDas that are accessible via the web (http://code.google.com/p/mydas/wiki/Tutorials). The tutorials are classified by their level of difficulty: Beginner, Intermediate and Advanced.

The first tutorial helps the user in the setup of the example data source. No programming is required at this point, as it is limited to installation and configuration of MyDas. The intermediate level tutorial guides the user through the common scenario of having a text file with some annotations that need to be displayed in DAS. In this case, the user is required to program the parsing of the file and mapping of it into the model. The third tutorial explores the case of obtaining data from a database system, which is a common way to store/access data in bioinformatics environments. This tutorial requires not only the ability to program in Java, but also to understand queries written in SQL.

The power of MyDas is revealed when it is used on large data sets with elaborate schemas as in the last tutorial. The example uses the freely available mysql database provided by Ensembl. There are over a hundred databases hosted on the Ensembl servers, and in this case we used the core set of tables for Homo Sapiens (version 56_37a), and restricted our search scope to some high level features (e.g. Chromosome, genes, transcript).

Many institutions may have a similar setup, however schema, policies and software vary from one location to another. Although exporting files (and using them to publish data) is an option, it implies that changes in the database won't be reflected in the generated files. In contrast, MyDas can be set up to take the information directly from the database management system and therefore will always be up to date.

A data source served by MyDas can be used by several tools to visualize its data. Figure 2.1.2 is a snapshot of the Ensembl browser, including the track named 'Ensembl Test', whose information is obtained from the tutorial data source. This demonstrates how the data published with MyDas can be displayed in well known genome browsers.

### 2.1.5 Other DAS Servers

The second tutorial mentioned above, describes the case where a user has a plain text file that follows a basic format (i.e. separated by a predefined character). This scenario is so common in bioinformatics

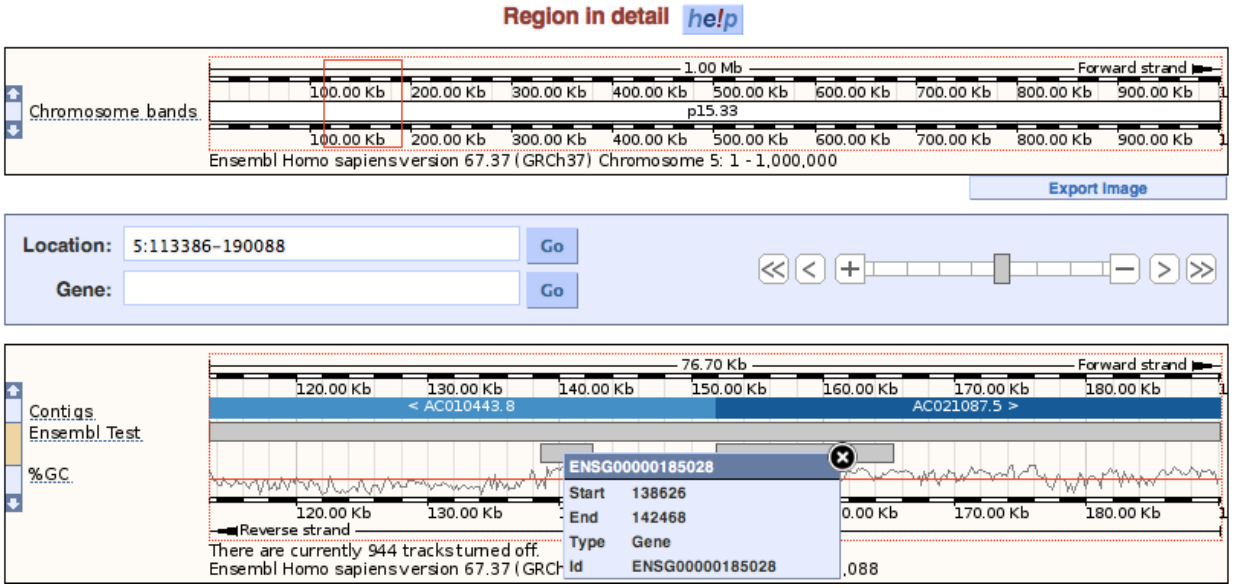


**Figure 2.1.2:** The data source created during the MyDas tutorials as it is visualised on the Ensembl web browser.

that a software tool specialising in the publishing of this type of data has been built. easyDAS[33] is a preinstalled server where a new data source can be configured by submitting a GFF file (or similar). This alternative is ideal for rapid publishing because it simplifies the tasks related to the hosting and storage of both data and server. The drawback is the lack of control over the data once it is deployed, because the owner of the data won't be able to change individual values of a dataset that has been submitted, instead, he would be required to delete and upload the whole data source.

Besides MyDas, there are other implementations of DAS servers such as Dazzle (`www.biojava.org/wiki/Dazzle`) or ProServer [26], amongst others (see `http://www.biodas.org/wiki/DAS/1#Implementation`). Nonetheless, MyDas and ProServer are the only servers that fully support the current DAS specification (1.6E). They differ from each other mainly in the language in which they are implemented (ProServer is written in Perl), but not in feature set, making system compatibility the major factor in deciding between the two.

Table 2.1.1 summarizes some of the high level characteristics of the most well known DAS servers.

Some benchmarking tests were executed in order to compare MyDas, ProServer and Dazzle and evaluate their loading performance. The tests were run with the Apache HTTP Server Benchmarking Tool (`http://httpd.apache.org/docs/2.0/programs/ab.html`). easyDAS was not considered for this test because it is installed on a different server, therefore there is no way to exclude network latency from the

[1]There is a branch of this project where the capabilities of DAS 1.6 have been implemented, however there was not a stable version of it at the time of publishing.

| *Feature* | *MyDas* | *ProServer* | *Dazzle* | *easyDAS* |
|---|---|---|---|---|
| Language | Java | Perl | Java | Web App(Perl) |
| Latest Release | 2011 | 2011 | 2010 | 2011 |
| DAS Version | 1.6 | 1.6 | 1.53E[1] | 1.6 |
| Physical Storage | Defined by User | Defined by User | Defined by User | Internal database |
| Entity Responsible | EBI | Sanger Institute | Sanger Institute | EBI |
| Main task to create a data source | Develop a Java class | Develop a Perl adaptor | Develop a Java class | Submit a tabulated file |

**Table 2.1.1:** Features of the main DAS servers

test.

The three servers were installed on the same machine, and data sources with the same data set were created on each. We prepared three DAS queries with expected responses of approximately 1.5Kb, 200Kb and 7.5Mb. Each query was repeated 1000 times with 10 concurrent connections

The 3 servers were able to complete all the requests and table 2.1.2 shows the main results of the executed test.

| *Figure* | *MyDas* | *ProServer* | *Dazzle* |
|---|---|---|---|
| Requests per Second - Mean (small) | 739.88 | 1.54 | 424.56 |
| Time per request - Mean (small) | 13.516 ms | 6492.978 ms | 23.554 ms |
| Transfer Rate (small) | 1534.68 Kbytes/sec | 2.81 Kbytes/sec | 859.91 Kbytes/sec |
| Requests per Second - Mean (medium) | 51.52 | 1.40 | 34.10 |
| Time per request - Mean (medium) | 194.114 ms | 7123.396 | 293.216 ms |
| Transfer Rate (medium) | 10944.96 Kbytes/sec | 288.19 Kbytes/sec | 6997.52 Kbytes/sec |
| Requests per Second - Mean (large) | 1.79 | 0.32 | 1.10 |
| Time per request - Mean (large) | 5589.148 ms | 30942.590 ms | 9110.292 ms |
| Transfer Rate (large) | 13088.38 Kbytes/sec | 2283.04 Kbytes/sec | 7770.29 Kbytes/sec |

**Table 2.1.2:** Benchmarking between the main DAS servers.

This comparison can not be considered conclusive in deciding which is the best or fastest DAS server, mainly because each data source has unique challenges and different cases can generate different results. Nonetheless, given that the three servers provide a data source implementation to publish data from a GFF file, they were configured to use the same GFF file to ensure equal conditions.

The figures in the table show that in all three scenarios MyDas performed better than the other servers. It is important to note that both MyDas and Dazzle were running on the same Tomcat server, therefore the conditions for both were the same. ProServer on the other hand, is a standalone server that implements socket communications in the application itself, which is an advantage in terms of making it easy to use.

The complete results of the tests are available in Appendix A.

## 2.2 DAS Writeback

A large part of the work included in this section was carried out prior to the timeframe of this PhD during the MSc referenced as No. 3 at the beginning of this chapter. Nonetheless, the inclusion of this contribution in this document is relevant because the software was reengineered as part of this PhD, upgrading it to the latest DAS specification and using the most recent versions of its dependencies: MyDas and DASty3. The results of both the MSc and reengineering process were published in the article 2 as reported at the beginning of this chapter. The first author of the paper is Gustavo A. Salazar and the co-authors provided input in line with their roles as collaborators and supervisors.

### 2.2.1 Overview

DAS offers a uniform protocol to share data that can, and has, been used by diverse data providers. One of the advantages of this approach is that users can now collect information from multiple resources by using a single tool. However we consider that opening the communication in the opposite direction (i.e. users submitting annotations to the resources), is a desirable functionality that was not included in the first versions of the protocol.

For example, researchers using a DAS client to gather information about a protein of their interest can identify errors in existing features or missing annotations. It is unrealistic to expect such a user to deploy a new DAS source to publish a handful of annotations. The traditional alternative is for them to report the new features to the providers. This however takes time, and has the inconveniences associated with using a centralised repository (as mentioned in section 1.1.1), where the user might have to wait for the release of a new version of the data before it can actually be used in the client.

As a solution to this problem, we have designed and implemented the Distributed Annotation System

(DAS) Writeback, which enables community-based manual annotation of public data. Our approach makes the process of manual annotation a collaborative task, whereby any individual can participate by sharing their knowledge in the form of new or edited annotations.

The DAS Writeback system provides the capabilities of reading, writing, editing and deleting features by users of a web application. For the design and development of such a system it was necessary to model an architecture that supports the new features, define an extension of the DAS specification to accommodate the client-server communication, and implement server and client components. All of these milestones were achieved while trying to follow the same style as the existing DAS technology,with the object of achieving easy adoption of the system by the DAS community.

### 2.2.2 DAS as RESTful services

The strategy used to define the architecture of a writeback system for DAS was to reconcile two technical concepts: on the one hand, DAS uses the REST protocol for web services [88], and on the other hand, reading and writing functionalities have been widely explored in Relational DataBase Management Systems (RDBMS), where the basic operations are known as CRUD (Create, Read, Update and Delete) [58].

One of the major strengths of the RESTful strategy is that it is based on such widely adopted standards as HTTP, XML, URI and MIME. This makes REST, and therefore DAS, technologies easy to implement and attractive to both developers and end users. Considering this we decided that when extending the DAS protocol to support servers that can store edited annotations, we set out to retain compatibility with the existing read-only system of HTTP GET requests.

One of the main features of the REST architecture is to have a uniform interface. This means that REST resources should be manipulated using a predefined set of operations. In the case of the Web, those operations are the 4 basic reading/writing operations CRUD, and the HTTP methods PUT, GET, POST and DELETE are suggested in the literature to specify those actions. These operations *"are broadly applicable but they also help uphold specific Web architectural properties"* [115].

Version 1.53E of the DAS specification includes a detailed explanation of the reading and querying capabilities of the system. This however only covers the use of GET, and there is no mention of any of the other RESTful methods.

Our proposal was to extend the DAS protocol in order to define the use of the other RESTful methods to complete the inclusion of the CRUD functionalities into the DAS system, and consequently, provide the DAS community with appropriate tools for collaborative annotation.

A previous effort in a similar direction was the object of a MSc thesis, where a DAS Writeback server was developed as a proof of concept [39], however it used different technologies and wasn't closely

attached to the RESTful protocol, which are factors that we consider to be key in the adoption of any extension on DAS.

### 2.2.3 Architecture

The components of the system were developed bearing the following goals in mind:

1. The original annotations of a DAS source should not be modified directly.
2. The system should be trusted by the user.
3. The system should promote interaction between the server and users.

The first goal has been established in order to ensure that the data shared by a provider is not at risk of being changed or deleted through any case of vandalism or by mistake when using the write back system. In order to do this, the architecture includes a third party writeback server that stores the changes to a set of annotations, independent of the original source providing those annotations.

New features, edits and deletions are saved on the writeback server as annotations themselves and therefore they can in turn be edited or deleted by other users. We hope that this feature gives the users the sense of trust (goal 2) that other collaborative systems such wiki projects have, where the community ensures the quality of the information.

Finally, as a means to promote the use of the writeback system, we implemented its functionalities in the well known protein DAS client: DASty. The purpose of this was to introduce the writeback functions to an existing interface that users were already familiar with.

Figure 2.2.1 represents the architecture of DAS including the writeback server. Firstly it is necessary to highlight that, when a feature is requested, the writeback server behaves as another annotation server, using the same DAS commands. The way this information is rendered is the responsibility of the client.

A standard DAS transaction starts by querying the DAS Registry (the DAS Registry provides a repository for the registration and discovery of DAS services). Next the reference sequence is obtained, followed by parallel requests to several annotation sources. The interaction between client and writeback occurs after the client has retrieved and displayed all of the information for the target protein, since it is only then that the user has a complete landscape view to take the decision to add, update or delete a feature.

HTTP requests relating to write operations on the writeback server are much larger than standard DAS requests (shown in Figure 2.2.1 as the width of the red arrow). The reason for this is that the client is now required to send the information to add or update a specific feature, including its type, category, position and other characteristics predefined in DAS. The communication with the writeback server is

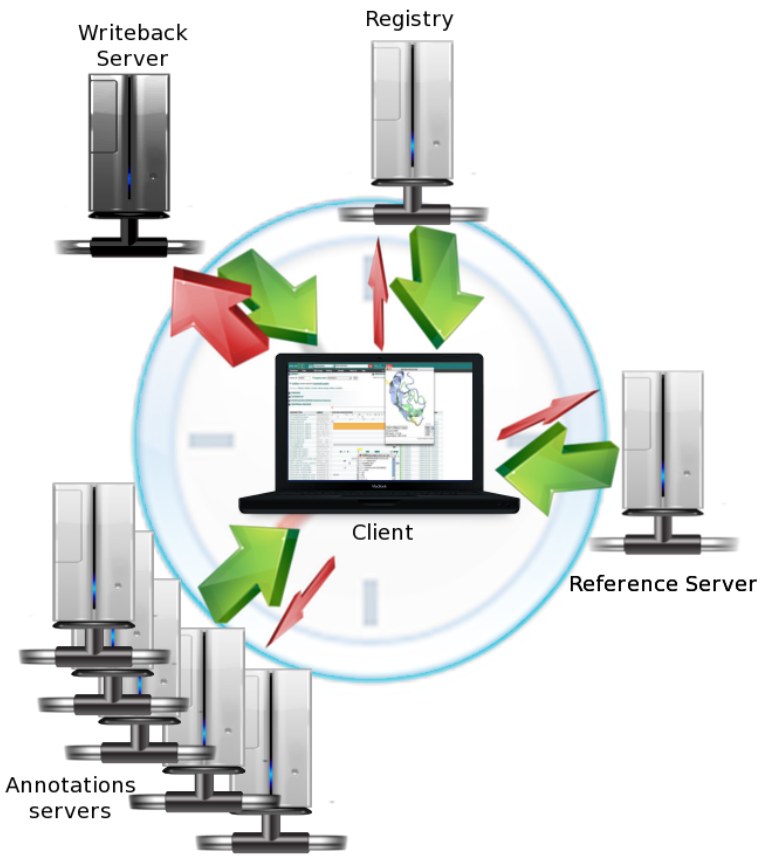


**Figure 2.2.1:** *Writeback in the DAS Architecture*: Extension of the DAS architecture for the writeback. A third party write-back server is the last to be queried by the client, and its response is used to update the information provided by the annotation servers. Communication with the writeback server has the peculiarity that the amount of information sent by the client is considerably larger than for any other server. The clock in the background represents the chronological order of the actions in a DAS transaction.

thus extended beyond the display of a graphic that compiles the information from all the servers. This is when the user starts to interact with the information, transforming the client from a pure visualisation tool to an interactive interface between the user and the DAS data.

### 2.2.4 Protocol Extension

As described in 1.1.2, version 1.53 of the DAS protocol was expanded in version 1.53E, where a set of proposed extensions were included. The same schema was followed when version 1.6 was released, shortly followed by 1.6E, when it was decided by the DAS community that new extensions could be proposed using the wiki page of the project (`http://www.biodas.org`). Their adoption by the community would determine their inclusion as a core component in future versions.

We followed these guidelines in order to include the writeback into DAS, which eventually became the first official extension of DAS 1.6E. The specification can be found on the DAS1.6E web page (http://www.biodas.org/wiki/DAS1.6E#DAS_writeback). It indicates that both input and output documents for the writeback should follow the DAS GFF format (See code below for an example). The HTTP method determines what to do with the received document; the GET method is still being used for querying and reading; while POST, PUT and DELETE should be used to create, update or delete features respectively.

The HTTP status codes used for DAS remain valid and indicate success or failure of the requested command (e.g. HTTP code 200 for success)

```
<?xml version="1.0" standalone="no"?>
<DASGFF>
 <GFF>
    <SEGMENT id="P13569" start="1" stop="1480" version="f29b8c0a9056a0f7680f3290d259b6ac">
       <FEATURE id="new" label="ISFCSQFSWIMPGTIK">
           <TYPE id="Polypeptide" cvId="Polypeptide">Polypeptide</TYPE>
           <METHOD id="ECO:0000160" cvId="ECO:0000160">
                   Inferred from protein separation followed by fragment identification
           </METHOD>
           <START>488</START>
           <END>503</END>
           <NOTE>note added in the writeback</NOTE>
           <NOTE>USER=username</NOTE>
           <NOTE>PASSWORD=password</NOTE>
       </FEATURE>
    </SEGMENT>
 </GFF>
</DASGFF>
```

### 2.2.5 Implementation

We have developed a DAS writeback tool by extending existing DAS clients and servers. The writeback is included as a plug-in of DASty3 and is integrated in the latest implementation of MyDAS. It is also compliant with the current DAS 1.6 specification.

*Server*

Our implementation of DAS Writeback is an extension of the MyDAS server, which has been widely described in section 2.1. A writeback data source was implemented to store annotations. Annotations are

the main entity in the data model, and any edits or deletions of an annotation are considered to be versions of the original annotation.

The datasource uses Hibernate [7] as its layer to access the persistence data, which brings the advantage of being *Database-Engine* independent. The data source has been successfully tested using PostgreSQL, MySQL and Derby; but is expected to work in other engines.

*Client*

One of the goals of this project was to create the perception for users that the writeback functions in a client are native and can be used naturally with existing clients. For this reason, the extension of an existing client was preferable to implementing a new one from scratch. In addition, the writeback server behaves as any other DAS server for reading purposes, so many software routines of an existing client could potentially be reused for the writeback visualisation.

We decided to use DASty for our first DAS writeback client and a prototype was developed for DASty2, which is detailed in [93]. We collaborated in the development of some of the components of DASty2 [50] and our knowledge of the software influenced the decision to use it for the writeback.

Later, in 2010, DASty2 went through a refactoring process, optimising its code to provide a plug-in framework. The new version is called DASty3 and our contribution to its development is described in section 2.3.1. The writeback client was rewritten as a plugin for DASty3 and is included in its core feature set.

*CRUD functions*

Below we describe how the CRUD functions (i.e. Create, Read, Update, Delete) have been implemented in both client (DASty3 plugin) and server (MyDas extension).

Create

Previous versions of MyDas used GET and POST methods interchangeably, therefore the first modification of the server was to identify the different HTTP methods and redirect the flow of data to its corresponding action. In order to create a new annotation, MyDas now listens to activity from any POST method, and if its content is a GFF file, it is parsed and added to the data source.

On the client side, a new form was added under the annotation graphic (2.2.2 B) in which the user can describe the new feature specifying label, location, type and evidence. Notes and links are optional fields for the new feature.


**Figure 2.2.2:** Writeback extensions on DASty3. (A) Modes to visualize the informations from a writeback server into the DASty3 features graphic. (B) Form to add a new feature. (C) popup window including writeback related tabs, for edit, deletion and duplication of features. (D) Annotations from the writeback server displayed in the features panel.

Version 1.6 of the DAS specification recommends the use of ontologies in order to standardise both annotation type and evidence codes, and make the task of integrating annotation from several servers easier. In order to promote the use of those ontologies, a list of suggested terms from the corresponding ontology is displayed in the edit form.

### Read

The writeback server behaves like any other DAS source when a set of features is requested. The client decides when and how to process this information. For the DASty3 writeback plug-in, the user has three different modes to choose from (Figure 2.2.2 A): (i) to ignore the writeback annotations, (ii) to use the writeback as any other annotation source, or (iii) to replace original annotations with the writeback information. The latter option generates a similar graphic for features as normally rendered by DASty3, but incorporating the modifications that the writeback server contains.

Update

When a DASty3 user clicks on a feature, a popup window with the complete information of the annotation is displayed. We have extended this functionality by including tabs on the popup window. The "Edit" tab contains a similar form to that of a new feature, but it is filled with the information of the selected annotation (Figure 2.2.2 C).

The information of this form is sent to the server using the method PUT. MyDas identifies the method, parses the GFF document and stores its content as the current version of the feature. The latest version will be the one to which the server returns for future requests.

Delete

Del is another tab available in DASty3 when a feature is selected. Its content is just a confirmation dialog where the user expresses the desire to remove that feature. The DELETE HTTP method is then invoked, including the feature's identifier. When MyDas detects a DELETE requests, it creates another annotation to register this command. This means that features are not really deleted from the server. Rather, the features tagged as deleted will be transparent in the DASty3 graphic, and only its border will be visible.

*Other writeback functions*

A basic module to allow for user authentication through a login and password was added in the writeback panel (Figure 2.2.2 A). Any writing function is conditional on prior login and password validation. The reading functionality does not require authentication.

Another way to edit a feature is through the history tab. In this window, the content of any previous version of the selected feature is displayed. The user can choose to roll-back to a previous version, which submits a new edit request through the PUT HTTP method. Therefore a roll-back task does not remove the existing history, but rather adds a new edit to it.

## 2.3 DAS Clients

The architecture of the Distributed Annotation System described in 1.1.2 passes a large part of the responsibility of integrating data to the client, whose job is simplified by the fact that all the sources comply with a protocol. Although DAS is mostly associated with the integration of information, DAS clients have been developing visualisation strategies in order to provide a single view of the collected information.

In the following subsections we will describe our contributions to two DAS clients: DASty and probeSearcher. Through these projects, we have observed the power of integrating information using

web based visualisation techniques. Users can visit the website of any of these tools wherever they are, and through the DAS client get information from multiple sources. This information can then be condensed into a single view, which can be explored further in order to get more detailed information.

### 2.3.1 DASty

Our first contribution to the DAS technology was in 2008, when we assisted in the development of a component for DASty2 which included the visualization of the 3D structure of a protein in parallel with its annotations, allowing the user to highlight the coverage of a feature in its structure. Later, as part of this PhD we collaborated on the reengineering of this software. The results of this collaboration are described in [114] and are also referenced at the beginning of this chapter as the co-author publication No. 4. Our contribution to the development of the software was focused on the software design and definition of the architecture. Implementation, documentation and maintenance were tasks executed by our collaborators.

DASty was originally developed using Macromedia Flash for the client and Java for the server component. It included the basic functionalities of an annotation viewer (e.g. alignment of annotations against the protein sequence, manipulation of the graphic) and was strongly attached to the UniProt DAS source [52].

The project was re-written in 2007 to take advantage of the novel features offered at the time by a technology known as Asynchronous Javascript And XML (AJAX). There are three main advantages of using AJAX for a DAS client:

1. The possibility of having asynchronous calls to a server allows the client to generate responses on-the-fly, and in this way the user can analyse partial results of fast servers, while slow ones take their time to reply.

2. The DAS specification uses XML to define the responses of a DAS server, and AJAX was designed to use XML for the same purposes. Therefore routines in the AJAX API can be reused for the processing of well-formed XML documents, such as the ones defined using the DASGFF format.

3. AJAX is part of Javascript and is therefore supported by all the main browsers. Besides that, the processed XML response can be used to manipulate the HTML rendered in the webpage. The whole system uses the technologies that are natively implemented in web browsers, and therefore there is no need for any third party libraries (e.g. Flash, Java.).

The interface of DASty2 organises its available functionalities as interconnected panels. In the *search panel*, the user can include the accession number of a protein of interest and select groups of DAS sources

to search. Once the search starts, multiple information messages are included in the *status panel*, and a progress bar indicates how many servers have responded. The first response to be displayed comes from the reference server, including the sequence of the protein which is then rendered in the *sequence panel*. The *positional features panel* contains the protein annotation browser, which gets updated by each response of an annotation source. These responses also include information that relates to the whole protein and not only to a part of it, which in DAS terminology are known as non-positional features. These features are grouped and displayed in the *non-positional features panel*. All these results can be filtered by type, server name and evidence code by using hierarchical lists that are displayed in the *filtering panel*. A last request is invoked to check if the protein has 3D structures reported in the PDB database. If this is the case, the user can select one and it will be displayed in the *protein structure panel* [51].

In the DASty3 project we re-engineered the software to not only offer the same range of capabilities as DASty2 but to also provide "*a framework upon which tools can be built and integrated into a cohesive web environment*" [114]. We defined a modular architecture (Figure 2.3.1), which supports the extension of DASty3's functionality by using plugins and templates.

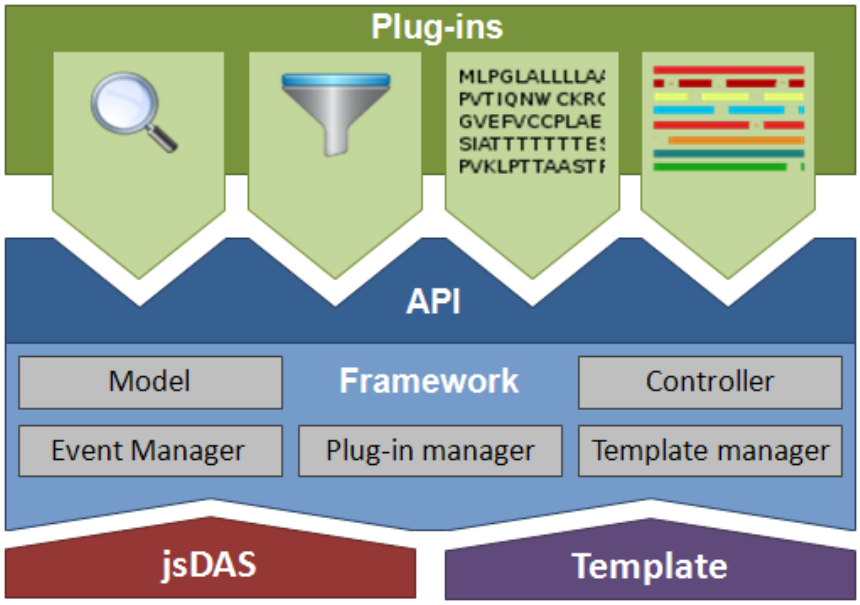


**Figure 2.3.1:** DASty3 Architecture

DASty3 templates allow the definition of styles through CSS, but more importantly, it supports manipulation of the complete interface through Javascript. This feature provides a way to adapt DASty3 to different environments, for example, if an institution decides to embed DASty3 into a web application, they can personalise the DAS client to not only follow the fonts and colours of the institution, but also to follow the same web layout of the application.

New features and visualisations can be included in DASty3 with the implementation of plugins. We defined a rich API that developers can use to interact with the different components of DASty3. For instance the interaction between plugins is done by implementing the broadcaster–listener metaphor using standardised events that can be used (to trigger or to listen) by any of the components in DASty3

All the communication between DASty3 and the different DAS entities is handled with jsDAS(`https://code.google.com/p/jsdas/`), a Javascript library that supports the DAS commands and parses the sources' responses into a data structure. This model is reused inside the DASty3 core and its plugins.

Figure 2.3.2 is a snapshot of DASty3, in which it is possible to identify the different panels that were mentioned above for DASty2. We decided to keep a similar "look and feel" for the application to facilitate its adoption by regular users of DASty2.

Figure 2.3.2 also illustrates the communication between plugins. When a user selects a track on the annotation browser (see the checkbox at the right side of the second track), an event notifying this action is fired, and several plugins react accordingly. For instance, the sequence plugin highlights the amino acids covered by the features on the track using the same colour as in the graphic, and simultaneously the 3D structure plugin highlights the corresponding parts in the current PDB file.

*Writeback plugin*

While developing the writeback plugin for DASty3 described in section 2.2.5, several software contributions were made to DASty3. Besides bug fixes and minor changes, there were a few significant improvements that are described below:

The communication between the client and the writeback server has some differences with respect to the communication with other DAS servers. Firstly, the different HTTP methods (PUT, GET, POST and DELETE) should be used according to their function. For this reason, the proxy component of DASty3 was extended to support the appropriate method usage.

The second difference is in the amount of information transferred: before the writeback, all the requests in DASty3 used the *GET* method. Therefore, the information sent from the client to the proxy was limited to 256 characters, which is the URL size limit for some web browsers and servers. With the writeback functionalities however, the client sends an XML document that is likely to exceed the URL size limit, making the use of other HTTP methods mandatory. This reinforces the applicability of the choice of adopting the RESTful standard.

An extension was added to the feature details plugin (which displays a popup window) in order to support tabs and include more than one page of information in the window.

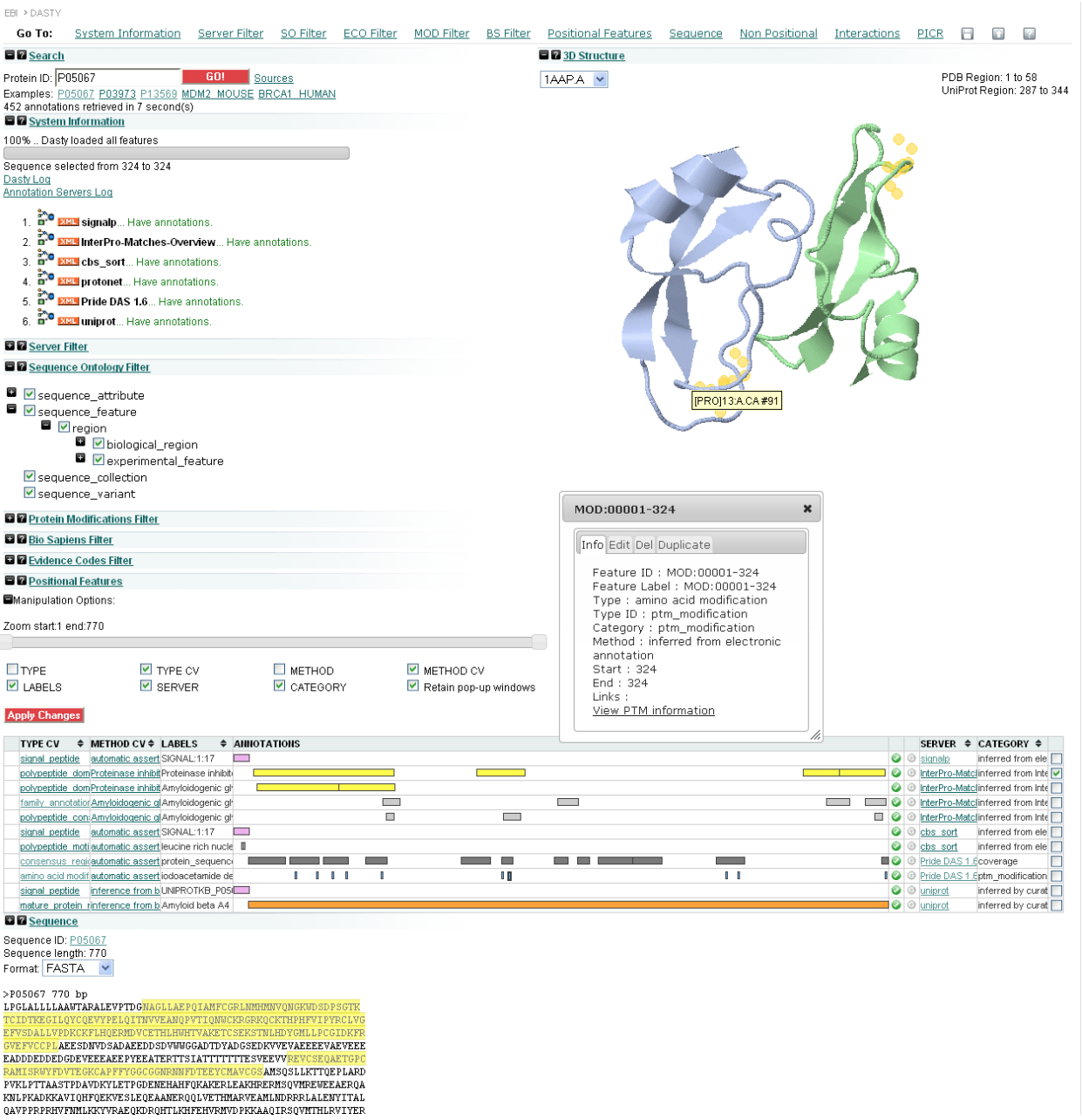


**Figure 2.3.2:** Snapshot of DASty3

### 2.3.2 probeSearcher

A microarray experiment is a way to quantify levels of gene expression in a sample. The methodology in these types of experiments is to prepare a grid of pre-identified sequences, called probes or oligos, where each probe is associated with a gene. The RNA sequences present in the sample are exposed to those in the grid, with the expectation that sequences that are complementary will bind to each other. The number of sequences attached to a particular probe is an indication of how often the target gene has been expressed in the sample.

A microarray chip is a high density grid of probes that has been prepared for an experiment with certain conditions. Because of the costs involved in the preparation of such a chip, vendors have fabricated chips that can be used in diverse scenarios, for example it is common to find microarrays with all the known genes of a particular species.

Microarray providers include documentation with their chips indicating which probes have been included and which genes or genomics regions they hybridise with. For scientists interested in gene expression, it is possible to find which protein or proteins can be detected using a microarray probe. However there is no easy way to get a list of available microarray probes for a specific gene or protein. There is also no centralised location which offers microarray probe information for several platforms and microarray chips.

We have developed a tool called probeSearcher (`http://biosual.cbio.uct.ac.za/probes-search/`) hoping to provide a solution to this issue. With probeSearcher, a scientist working on a set of proteins or genes can find out which microarray chips include probes associated with them. This information helps the researcher to find the right chip for their project, and to avoid expending unnecessary resources.

ProbeSearcher has a minimalistic interface (see figure 2.3.3), where the user is only required to input the accession numbers of interest. The default search assumes the user has UniProt IDs and is looking for the probes and ultimately the chips that include them. The search can be modified to use Ensembl IDs or inversely to find out which genes or proteins are associated with a given probe ID. The tool also offers some advanced search options, which allows the user to filter the search through different vendors and/or the target organism.

The result of a search with probeSearch is a list of the probes linked with each of the proteins queried, including a link to the Ensembl browser where the location of the probe is shown in the context of the chromosome that contains the target sequence.

Results can also be grouped by chip, which is especially useful in the case where the query consists of several proteins, because in this way it is possible to see which proteins are represented on the same chip. For example, it is evident from the results shown at the bottom of the figure 2.3.3, that the chip

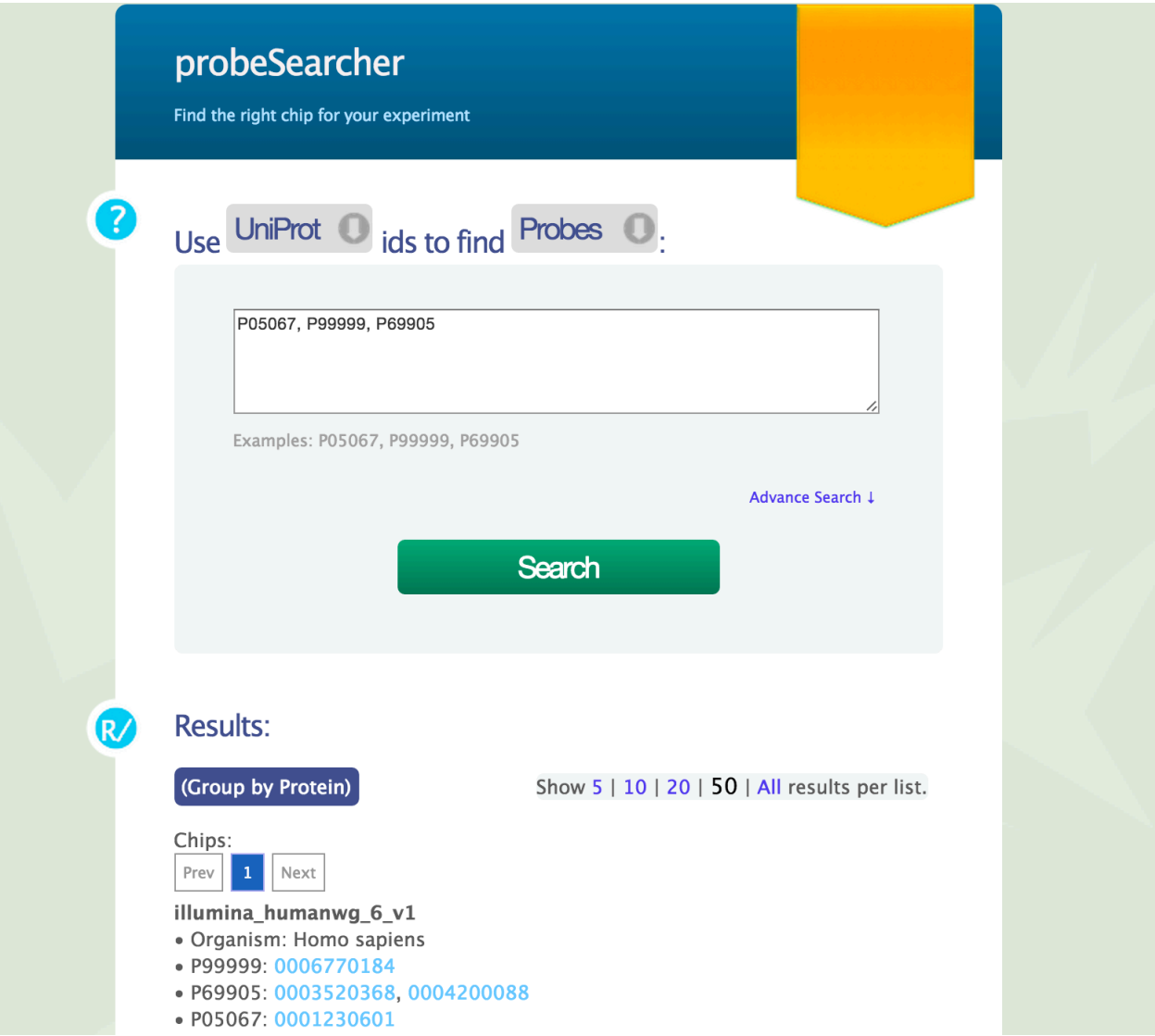


**Figure 2.3.3:** Snapshot of probeSearch. The query includes three UniProt accession numbers: P05067, P99999, P69905. The result has been grouped by chip, and the first hit is shown at the bottom.

*illumina_humanwg_6_v1* contains probes associated with the three protein of interest.

*Implementation Details*

We have used DAS in order to support the functionalities of probeSearcher, creating two datasources in MyDas to store and share the connections between: (i) UniProt IDs and probes (uniprot2probes); and (ii) Ensembl IDs and probes (ensembl2probes). Using DAS for the development of this tool not only gives us the advantage of reusing robust software for the client-server communication, but also converts the data into a sharable format that can be used in other tools. For example the uniprot2probes data source can be added as another source into Dasty3 in order to include the list of related probes from several platforms.

The data stored in these sources are taken from the Ensembl BioMart [59]. We have developed biomartProbes, a command line tool in java that queries BioMart, creates temporary files with the info, and uses those files to load the information into a Solr server. The DAS sources dynamically query the Solr server whenever a request is received. The source code of biomartProbes together with a packaged version can be found at `https://github.com/4ndr01d3/biomartProbes`.

The client of this system is a web application available in `http://biosual.cbio.uct.ac.za/probes-search/`. It has been written in Javascript, HTML and CSS, and takes advantage of the features provided by AJAX (described in 2.3.1). The code of this web-client can be obtained at `https://github.com/4ndr01d3/probeSearcher`.

*DAS extension for advanced search*

While developing the advanced search functionality of probeSearcher, we identified limitations in the way that DAS filters the responses of the queries triggered by the *features* command. DAS queries are carried out by the *features* command, which allows the querying of annotations within a segment. The user should input the segment ID and optionally a range of coordinates and the server responds with the features that match the parameters. The response can be filtered by *type ID* or *category ID*, however this requires that the user knows all these IDs, and therefore text-based searches are not supported.

For this reason we proposed a new extension to the DAS protocol (`http://www.biodas.org/wiki/DAS1.6E#DAS_search`) that adds the optional argument *query* to the *features* command. The content of the new command uses a syntax based on the Lucene query language, and allows the filtering of the results using the elements and arguments of DASGFF files.

The strategy behind the DAS advanced search extension was inspired by the methods used in PSICQUIC (Section 1.1.1), where a similar query language called MIQL was defined in order to support advanced search options to servers containing molecular interaction data. The Lucene query language

(http://lucene.apache.org/core/2_9_4/queryparsersyntax.html) and sublanguages based on it, such as MIQL or the one used in probeSeacher, were created in order to facilitate the exploration of data in indexed files, where all the information is stored in data structures that can be seen as a single table with many columns (Fields) and rows (Documents).

Listed below are some of the most useful features of the syntax of the Lucene query language.

- Text values are represented in between quotes and called phrases, and a phrase is composed of several terms, all of which are single words.
- A query can be defined by indicating which value is required in which field, using the notation of the name of the field followed by colon ':' and then the value. e.g. *uniprotID:"P05067"*.
- Multiple queries can be combined with boolean operators: AND, OR and NOT
- Wildcards can be used within single terms. '?' represents any single character in that position, and '*' for multiple characters. e.g. *uniprotID:"P*9"* looks for documents where the value of the field uniprotID can be of any length but should start with P and finish in 9.

The specification in http://www.biodas.org/wiki/DAS1.6E#DAS_search contains details on the fields defined for the DAS advanced search, and the way it should be used in the *features* command.

We have implemented the new extension in MyDas, including the Lucene engine as a dependency. The support for an extra command called *indexer* was added to MyDas, which should be used to start the processing of any datasource, creating Lucene indexes in order to be able to give quick responses when the *query* argument is used. The code of this extension is included in the MyDas repository https://code.google.com/p/mydas/.

## 2.4 Discussion

DAS has proven to be a successful method to publish, share and integrate data, for example, its registry reported more than 1500 data sources by the end of 2014, including important providers such as the Ensembl project and the UniProt Consortium. Several providers implemented the protocol within their own tools, in order to include third party data in the context of their own. We consider that our contributions to the different projects described in this chapter were significant to the growth of DAS, but most importantly they can improve the daily routine of scientists.

MyDas currently forms the basis for high volume DAS servers like UniProt and InterPro. It combines performance and stability with ease of installation, operation, and extension. While the easyDAS server provides a platform for DAS-based sharing of small sets of nucleic acid or protein annotations, and ProServer addresses Perl-based environments, MyDas offers a developer-friendly solution for

laboratories and institutions that wish to share medium to large scale datasets in a Java-based environment. It completes the landscape of modern, open source DAS servers available to organisations sharing biomolecular data via the distributed DAS protocol.

The writeback specification is now an official extension in DAS and is considered to be a part of the core protocol. The developed software has been well received by the community and the server implementation is now part of the official development of MyDas. In addition, the client is included in the set of plugins of DASty3, which is a widely used DAS client. Despite this, the success or failure of any collaborative system is recognised through the interaction of real users with the system, and additional time is required to determine this. We hope this system will contribute to the creation of a more publicly accessible, easily updatable, and reliable protein knowledge base.

DASty has evolved during its 10 years of existence, from a Flash widget that was highly attached to the UniProt DAS server and had a limited set of options, to a very modular and extensible framework. DASty3 is developed with modern web technologies, and has a plugin-based architecture that can be operated as an application on its own or embedded in existing websites.

Lastly, the probeSearcher tool presents a visually appealing solution to the problem of finding the available microarray chips that contain probes associated with a gene or protein of interest. This project exemplifies the potential of the DAS protocol to integrate data from different domains.

Unfortunately the DAS technology has lost momentum in recent years and the community meeting that used to take place annually hasn't occurred since February 2012. Since then, the pace of adoption of the technology has been significantly reduced.

In August 2014, the EBI released an announcement about the retirement of the DAS services supported by the institution: http://www.ebi.ac.uk/about/news/service-news/das-services-to-retire. The justification behind this decision was the low level of usage of the services and the cost required for support and maintenance. It is also mentioned in the article that many of the EBI resources are currently supporting alternative data integration technologies. The support of some of the DAS services will be maintained by the institution until the end of 2015.

A shorter announcement was also included on the web site of the DAS registry: http://www.dasregistry.org/, indicating that the registry website will be discontinued on the 1st May 2015. We could not find any statements regarding the support of DAS services on either the Sanger Institute's webpage or that of the Ensembl project.

It is our opinion that the DAS protocol goes further than isolated REST services, particularly on the standardisation of queries and responses. This is by no means a criticism about the release of new REST services. For example, the potential of the recently released UniProt programatic access (http://www.uniprot.org/help/programmatic_access) is undeniable, and its functionality goes further than

what a regular DAS source can offer. However, the lack of a query language and response format that is common to other services forces the user to develop extra components to support each particular implementation of REST services.

We consider that withdrawing the support to DAS technology by these institutions is a step backwards for the integration of bioinformatics data, especially when it is not clear if there is a common strategy for publishing and sharing data for the services that have decided to move away from DAS. Also, we believe that the development of programmatic access in some of the providers should not be seen as a replacement of the services offered by DAS, because no matter how powerful a single one of those services is, it won't be able to include the combined data that can be aggregated by including all the data sources.

*Our posturings, our imagined self-importance, the delusion that we have some privileged position in the Universe, are challenged by this point of pale light.*

Carl Sagan, Pale Blue Dot, 1994

# 3

# Visualization in Bioinformatics

**First author publications** :

1. Gustavo A. Salazar, Ayton Meintjes, and Nicola Mulder. Ppi layouts: Biojs components for the display of protein-protein interactions [v1; ref status: awaiting peer review, http://f1000r.es/2u5]. *F1000Research*, 3-50, 2014
2. Gustavo A Salazar, Ayton Meintjes, Gaston Mazandu, Holifidy A Rapanoël, Richard O Akinola, and Nicola J Mulder. A web-based protein interaction network visualizer. *BMC bioinformatics*, 15(1):129, 2014

**Coauthor publications** :

3. John Gómez, Leyla J García, Gustavo A Salazar, Jose Villaveces, Swanand Gore, Alexander García, Maria J Martín, Guillaume Launay, Rafael Alcántara, Noemi Del Toro Ayllón, Marine Dumousseau, Sandra Orchard, Sameer Velankar, Henning Hermjakob, Chenggong Zong, Peipei Ping, Manuel Corpas, and Rafael C Jiménez. Biojs: An open source javascript framework for biological data visualization. *Bioinformatics*, 2013. URL http://bioinformatics.oxfordjournals.org/content/early/2013/02/23/bioinformatics.btt100.abstract

4. Manuel Corpas, Rafael Jimenez, Seth J Carbon, Alex García, Leyla Garcia, Tatyana Goldberg, John Gomez, Alexis Kalderimis, Suzanna E Lewis, Ian Mulvany, Aleksandra Pawlik, Francis Rowland, Gustavo A. Salazar, Fabian Schreiber, Ian Sillitoe, William H. Spooner, Anil S. Thanki, José M. Villaveces, Guy Yachdav, and Henning Hermjakob. Biojs: an open source standard for biological visualisation–its status in 2014. *F1000Research*, 3, 2014

**Author's Contibutions** :

1. Critical revision of the manuscript for important intellectual input: GS, AM and NM. Supervision: NM. Study concept: GS, AM and NM. Software development: GS. Drafting of the manuscript: GS and AM. All authors have read and approved the final manuscript.
2. Critical revision of the manuscript for important intellectual input: GS, AM, GM, HR, RA and NM. Study concept: GS, AM and NM. Software Design: GS, AM, GM, HR, RA and NM. Software development: GS and AM. Creation of datasets: GM, HR and RA. Software Testing: GM, HR, RA and NM. Software Documentation: GS and RA. Drafting of the manuscript: GS, HR and NM. Supervision: NM. All authors read and approved the final manuscript;
3. All authors have participated in the development of the BioJS community through provision of code, meeting attendance or writing of grants.
4. All authors have participated in the development of the BioJS community through provision of code, meeting attendance or writing of grants.

*A single point in a photograph can represent our whole known world*, as shown in the famous image acquired by the Voyager 1 spacecraft in 1990. From that perspective, is impossible to perceive all details that conform our planet, from rivers to highways, from mountains to buildings, even countries or full continents and oceans are undistinguished in the mentioned picture. Nonetheless, the image gave us a insight of the vastness of the universe in comparison with our known world.

As in the previous example the same object can be seen from different perspectives. Each of them can highlight some features and hide others, hence the importance of choosing the right representation for the object in display.

This chapter describes the contributions that are the object of this PhD project on the visualisation of bioinformatics data. We first described BioJS, a community project to create a library of bioinformatics web components, including some widgets that we have been developed to be part of the library. The second part of this chapter focuses on PINV, a tool to visualise PPI networks using web technologies.

## 3.1 BioJS: A JavaScript framework for Biological Web 2.0 Applications

We participated in the community effort to create a specification and develop an standard for web components in Javascript, called BioJavaScript, or BioJS for short. BioJS has been described in the publications 3 and 4 listed at the beginning of this chapter. We believe relevant to include a description about BioJS, not only because of our contribution to it, but also because we have followed the proposed standard for the creation of a set of visualisation components described in the section 3.1.2. Some of those components were used to create the web based tool PINV, described in section 3.2.

We want to explicitly express that by including this description here we are not taking credit of the development of BioJS. BioJS is the result of a group effort and has many contributors, most of which are listed here `http://biojs.net/biojs_team.html`. Our contribution to the project has been in the form of software, documentation and participating in multiple project meetings.

BioJS is an open-source and community-driven project that aims to provide a framework to facilitate the reutilisation of JavaScript components for the visualisation of biological data in the web [38]. At the hearth of BioJS 1.0 is a registry of components (`http://www.ebi.ac.uk/Tools/biojs/registry/index.html`) in which developers following the BioJS guidelines can publish their components; and simultaneously creators of web content for biological tools, can find the right visualisation widget for their purposes.

The project started at the EBI in 2011 and its first version was officially released when the first publication about the project was written [38]. By then, the registry had 29 component registered, thanks to an effort from all the collaborators in order to count with a heterogeneous set of tools available for this release.

In the first version a BioJS component should follow the object oriented paradigm and inherit from a common class called BioJS. In this class certain routines were implemented in order to ensure a common behaviour in between the components. The most important things included in this class were: an event manager, an object creation subroutine, an standard way to receive parameters and a group of utility functions.

An example of how several BioJS components can interact is shown in the right side of figure 3.1.1. The secondary structure of a protein (bottom-left) can be displayed when the protein is selected from a PPI network (top), and a region on the 3D structured (bottom-right) can be highlighted, when the mouse hovers on one of the substructures of the bottom-left component.

BioJS components are flexible enough to manipulate web elements for visualisations: SVG, Canvas, CSS, and in general anything that can be controlled via Javascript. BioJS is framework agnostic, allowing each component to define its own dependencies. This schema supports the interaction of two or more components that have been implemented using different JavaScript frameworks (e.g. JQuery, YUI, Prototype, etc.).

Another aspect defined in BioJS 1.0 was the use of JSDoc to include in-code comments, this code was not only use to create the usual reference documentation, but it was key in the generation of the running examples of the components in the registry.

The developer was requested to include snippets of its code and a list of the dependencies as part of the documentation.That information is used to generate the pages in the registry that include a running example, instructions on how to include the dependencies, and all the generated documentation for methods, events and parameters.

The layer separation in the BioJS architecture is shown in the left side of figure 3.1.1. The final representation of a component can be deployed in any of the different methods supported in web, the control of such representation is programmed in JavaScript, supported by any of the libraries the developer choose to have as dependencies. If the chosen representation is based on HTML standards, all its elements can be stylised via Cascade Style Sheets CSS [16].

Here are some examples of different methods to represent data in web with examples of BioJS components using them.

**HTML documents** Where HTML is generated to display data, for instance *Biojs.InteractionsTable* (`http://www.ebi.ac.uk/Tools/biojs/registry/Biojs.InteractionsTable.html`) creates a HTML table that contains information about protein interactions (Figure 3.1.2 a).

**Visualizations using HTML elements** Similar to the previous one, but the HTML generated is organise in certain way to represent the data. For example, *Biojs.Chromosome*

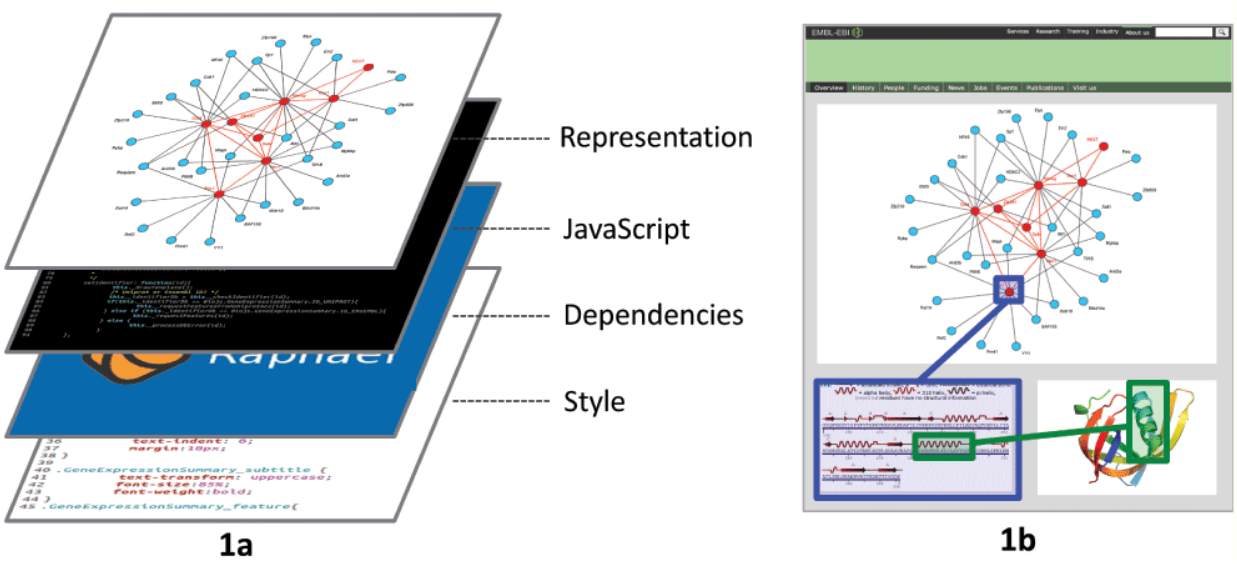


**Figure 3.1.1:** 1a shows the different layers that a BioJS component is divided into. The representation layer sits on top of the JavaScript layer, which similarly possesses a layer of dependencies and a style. 1b presents an example of interactivity between three components, a protein-protein interaction network viewer, a secondary structure viewer and a tertiary structure viewer.

(`http://www.ebi.ac.uk/Tools/biojs/registry/Biojs.Chromosome.html`) uses the div HTML element to create boxes representing the bands of a chromosome (Figure 3.1.2 b).

**Scalable Vector Graphics** HTML supports the SVG format, and because SVG is a Markup language, its manipulation is similar to when dealing with HTML content. The *Biojs.FeatureViewer* (`http://www.ebi.ac.uk/Tools/biojs/registry/Biojs.FeatureViewer.html`) uses this technique to display protein annotations (Figure 3.1.2 c).

**HTML canvas** Since HTML5 the element canvas is part of the specification, this component provides a programatic way to generate graphics. It is faster than SVG, but doesn't offer as much control on each of the drawn elements. For example, a representation of the metabolic pathways provided by KEGG is displayed by the *Biojs.KEGGViewer* (`http://www.ebi.ac.uk/Tools/biojs/registry/Biojs.KEGGViewer.html`) component using the canvas element (Figure 3.1.2 d).

**Browser plugins** The use of third party tools such as Java Applets or Adobe Flash objects is also possible as long an interface with JavaScript is provided, for example *Biojs.Protein3D* (`http://www.ebi.ac.uk/Tools/biojs/registry/Biojs.Protein3D.html`) displays the 3D structure of a proteins by using the JMol Java applet (Figure 3.1.2 e).

Besides the diversity in the chosen technology to represent the data, figure 3.1.2 also shows the variety

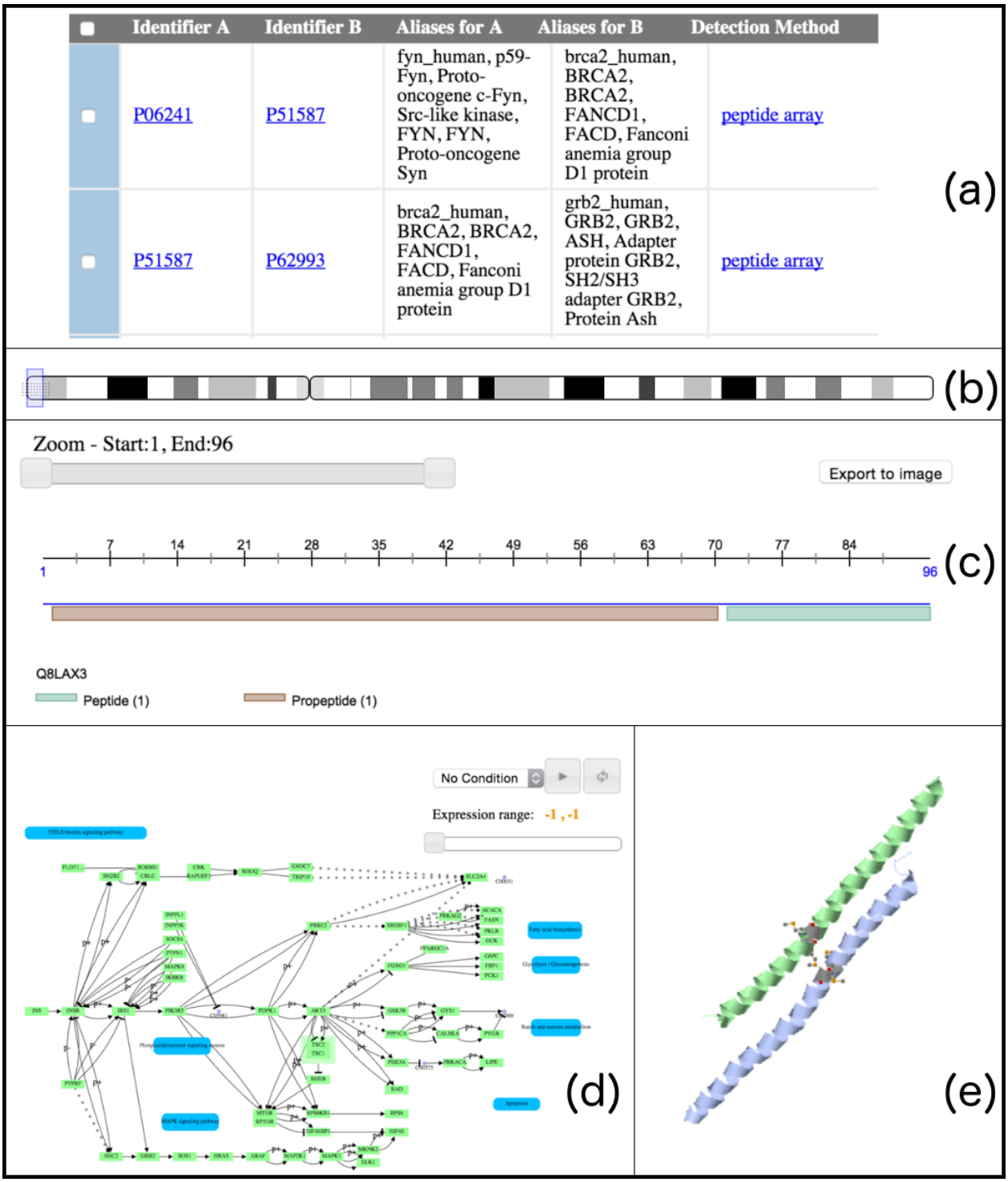


**Figure 3.1.2:** Snapshots of some BioJS components. (a) Biojs.InteractionsTable, (b) Biojs.Chromosome, (c) Biojs.FeatureViewer, (d) Biojs.KEGGViewer and (e) Biojs.Protein3D.

of visualisation techniques that can be implemented using BioJS, from text tables to 3D objects.

BioJS ambitious vision is "*that every online biological dataset in the world should be visualised with BioJS tools*". In order to accomplish it, BioJS requires to provide a convenient environment for the different roles related to the visualisation of biological data. The roles that we have identified are: data providers, developers of web components, developers using ready-to-use components and the users that finally visit web tools that include those components.

A data provider can benefit from BioJS by simply reusing existing components that can visualise their type of data, and in this way save the resources that a development form scratch requires. Even if there is not component that fully covers the needs of the institution, is likely that there is a component that provide a partial solution, and thanks that BioJS an open-source project, the institution can extend the existing component, improving its feature set, or create an alternative version of it tailoring to the needs of the institution. In the case where the required representation is so unique, that the institution requires to develop the component from scratch, they still get the benefit of ensuring that anyone else who will display their data, will count with a widget that visualise it exactly in the way they intended to do so. The two last cases ultimately reflect in growth over the BioJS library for the benefit of other BioJS actors.

Developers creating new visualisations with BioJS can get exposure and acknowledgements for their efforts. Their widgets will be now visible in an open forum, where a community of potential users and collaborators can both, benefit and contribute to the improvement of the widget. On the technical side, a developer following the BioJS guidelines is not limited by any dependency, and although the need of writing documentation and applying a code structure, implies more time on the creation of the resource, it also ensures the robustness of the components.

When a developer wants to include a component to display some type of biological data, they can take advantage of BioJS by easily follow the common installation instructions provided in the registry, which have been customised for each of the components. The registry supports navigation and search of components, but we consider the bigger benefit for developers is to be able to run live demos of the components and the code to generate them. In this way, someone interested in a widget can seen it in action, without having to install or develop anything.

All of it has as the final beneficiary the researcher, who ultimately is the one who will interact with the different components getting a dynamic view of the data in the way the provider expected, in a component that can be improved by a community of visualisation experts, and has been chosen to be in the web tool that the user is visiting because it is the one that highlights the features of the data in the best way.

We are aware that this is still a work in progress, but the results so far obtained from the project are very positive. For example, the journal F1000research decided to publish an special collection of articles

describing BioJS component. We have discussed before our opinion about how positive is for projects to provide an environment for the publication of peer-reviwed articles (Section 3.1.2). We consider that this collection of BioJS articles is an important step in the right path for the growth of the project and its contributions to the research community. The section 3.1.2 describes the details of an article included in this collection, in which BioJS components for the visualisation of PPI networks are introduced.

### 3.1.1 BioJS 2.0

Besides the relative success of the first version of BioJS, its community identified a number of deficiencies and areas where the project can be improved. For example after BioJS first version was released, the focus of the community was directed to new components, and for around 3 years the core did not change much. During that time some of the libraries and strategies used got outdated, for example, the registry required compilation using maven (`http://maven.apache.org/`) in order to generate the web content for each of the components. This seemed a good idea at the beginning of the project, however became a problem because of the delay in the publication of components, and more importantly the lack of a protocol to retrieve the dependencies of each components.

The registry was still bringing visibility to the components, but it was failing to attract developers to create new widgets using the proposed guidelines. This was the main motivation to work on BioJS 2.0.

A concentrated effort looking to push forward the new version of BioJS was held during the 4th and 6th of August of 2014. The event was hosted in Munich Germany (`http://biojs.net/code/2014/07/04/announcing-hackathon.html`) and it was also possible to collaborate remotely.

The motto of this event was to bring "easiness" back to BioJS. This principle has transcend from the event to become the goal of BioJS 2.0. The efforts were mainly made from the point of view of the developer of new components: BioJS should be easy to develop, maintain and test, which should be done without failing to offer the benefits to the other BioJS stake holders and therefore BioJS should continue to be easy to use, combine and discover.

The main strategy to be able to provide the desired "easiness" was to give freedom to the developer, which in BioJS terms meant to deprecate the predefined class where all the components used to inherit, it also implied to abolish the requirement to follow the JSDoc format to document and introduce easily editable, working examples of an component and moving the documentation to a public README file or any other form chosen by the developer (e.g. github wiki).

The alternative was to define a set of guidelines called "the gold standard" (`http://edu.biojs.net/series/102/70_gold_standard.html`). It includes recommendations on how to test, document, publish and create examples for a BioJS component. The key behind these recommendations

is that all of them are supported by web resources that can be accessed programatically. The registry is then able to detect when a recommendation is followed by a component, and present this information to users exploring the components. For example, the sniper package (https://github.com/biojs/sniper) facilitates the inclusion of live examples in the component's page.

This implementation in the registry was done by extensively use two well know development resources available in the web: The web-based Git repository hosting service: GitHub; and the node package manager (npm).

All the code of BioJS was migrated to the source control system git (http://git-scm.com/) and the repositories are hosted by GitHub (https://github.com/biojs/). The source code of the components is now independently manage by their authors, and the only components in the BioJS repository are the ones related with the core functions, such as the registry or helpers to follow the gold standard.

A developer creating a BioJS component should publish its code as a npm package (https://www.npmjs.com/). npm is in itself a registry of packages for javascript components, which has programmatic access from a command line client, but also can be browsed using its online front end (https://www.npmjs.com). All the configuration and metadata of a npm package should be included in a file named package.json, which should be located at root of the package folder. BioJS builds upon this standards and added custom fields for meta information to the package.json file.

By February 2015 npm had more than 120000 packages in its repository, with an extraordinary growth rate of 221 packages per day, as reported by the independent website http://www.modulecounts.com/. The high adoption of the npm repository is an advantage for BioJS developers because it provides an extensive tool set to work with, which scope goes beyond the visualisation of biological data.

With this model, the developer is now free to use any package to accomplish any of the BioJS guidelines, and as long it is reported in the package.json file it can be reported in the BioJS registry. The reporting of the gold standard implemented features it is still a work in progress, nonetheless a fully operational registry for BioJS2.0 is available at http://biojs.io/.

It is important to highlight the continuous effort from the BIoJS community to provide up to date documentation . The web portal http://edu.biojs.net/ contains tutorials and reference documents to help the developers of new BioJS components.

Figure 3.1.3 represents the development stages of a BioJS component, below we describe some of the technical details on how BioJS 2.0 assists in each of these stages.

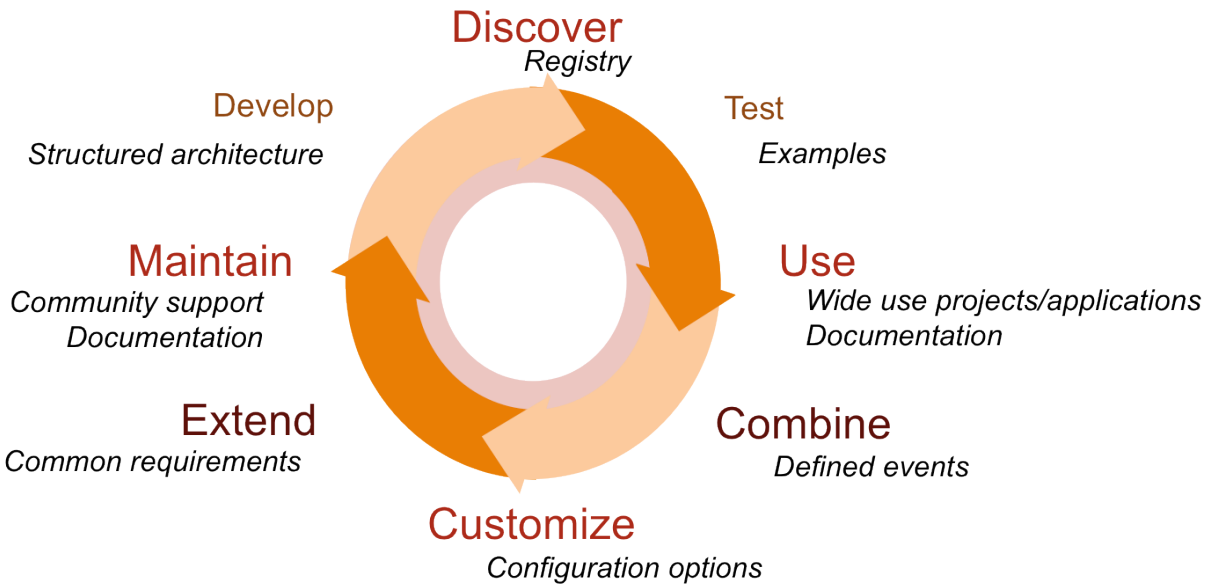


**Figure 3.1.3:** Cycle of a BioJS component.

*Develop*

The developer is free to use any javascript framework or visualisation library as a dependency. the only requirement is to report them in the package.json file. Ideally the dependencies should also be npm packages, in order to support recursive checking. However is also possible to include required files as part of the package.

There is an assistant package to generate a template for a BioJS project (`http://biojs.io/d/slush-biojs`). The package was created using Slush(`https://slushjs.github.io`) and once executed, it includes some dependencies and configurations useful to comply some of the guidelines in the BioJS gold standard.

*Discover*

The only requirement for a npm package to be included in the BioJS 2.0 repository is to contain the tag "*biojs*" in the package.json file. The BioJS registry queries npm to check for published packages with the "*biojs*" tag and processes their configuration files in order to extract all the information about documentation, tests, etc.

The developer is free to use other tags in order to better describe its component. Anyone interested in a visualisation component for biological data, can visit `http://biojs.io/` and then search by keywords, check what are the most popular components or the most recently updated. Each component page contains the documentation that the developer has included.

*Test*

Users interested in a component can try it out in the same registry page, if an snippet of the code was reported by the developer. The registry also provides links to execute the examples through online javascript editors such as JS Bin (http://jsbin.com/) or codepen (http://codepen.io/), which allow the user to edit the sample code to edit parameters and basically to have a playground area to test out the component.

On the developer side, the use of npm supports the execution of unit tests through several frameworks such as mocha (http://mochajs.org/), qunit (http://qunitjs.com/) or jasmine (http://jasmine.github.io/). All of which are available in the npm repository; thanks to this, all the unit tests can be ran by the execution of a a single npm command.

Freely available web resources such as travis(https://travis-ci.org/) or drone.io (https://drone.io/) can be setup to automatically run the tests every time new code is pushed to the github repository of the component. If provided, this information can be used by the BioJS registry when creating the component's page. In this way, explorers of biological components will know if the latest version of the code have passed all the unit tests.

*Use*

The simplest way to use a BioJS 2.0 component is to install it through the npm command line tool:

```
1 npm install <package_name>
```

To be able to run this command, the npm tool needs to be installed and it should have internet access. If this is the case, the components will be downloaded together with all its dependencies and extras included in the package (e.g. snippets and docs).

The developer can include other commands in the configuration of the file that can be executed once installed, for example to run tests, generate documentation, built minified versions of the code, etc.

It is also possible to configure a compiled version of the component that includes all the dependencies in a single file and serve it using a Content Delivery Network (CDN), which then provides a way to use the component on any web page by only including a single line in the header of the HTML. The author of a BioJS component can use the npm package browserify-cdn, which is a convenient way to publish via CDN.

*Combine*

Several BioJS components can be included in the same web page and displaying them simultaneously. However this method does not allow interactivity between the components, and to accomplish that,

some programming in Javascript is required.

As describe before npm provides support to handle dependencies, therefore to combine BioJS components, it is possible to write another npm package that declares the desired components as dependencies. Moreover, thanks to some tools such as CommonJS (`http://www.commonjs.org/`), Browserify (`http://browserify.org/`) and RequireJS (`http://requirejs.org/`) it is possible to declare when a component is required using JavaScript code. This provides a way to built a unified javascript that contains all the components and the code to integrate them.

The dynamic interaction between components can be reached through the broadcasting of events from one component, and the reaction to it form another one. As mentioned before, BioJS 2.0 removed the parent class where the uniform event handler was implemented, however a recommended BioJS component called biojs-events (`http://biojs.io/d/biojs-events`) is now provided to supply this need.

*Customize*

Each BioJS component defines the parameters need it to run an instance of it. The input of such parameters can be defined in several ways, for example, as attributes of the constructor of a JavaScript object, or as a configuration file, or even as HTML parameter on the element where the component will be included.

This mean that the level of customisation and the methods to do it are responsibility of the developer of the component. However we consider that is common understanding from BioJS developers that the ability of personalise a component it is an important feature and its documentation should be included as part of the package.

*Extend*

Since BioJS 2.0 there is not common repository for all the components, this decision, delegates the responsibility to manage the code that is been published through the repository to the developers. And it also allows the use of source control repositories such as GitHub, in which any developer can obtain the source code of the component, extend it and request the changes to be included in a future version.

This strategy gives the control of contributions, and extensions to the owner of the component, while liberating the members of the BioJS team from tasks related to that administration of particular components.

Alternative a developer can always copy an open source component, what in GitHub terminology is known as "*fork a repo*", extend it to implement new features, or to use it with a different purpose of the original developer, and then create another BioJS component from it.

*Maintain*

The maintenance of a component is again responsibility of its author in BioJS 2.0. By supporting this process with GitHub the developer can take advantage of the multiple tools provided by this service. For example, *branching* to experiment with the code without affecting the current release, or to have a list of *issues* where users can report bugs or suggestions, or even receive improvements from other users directly in the code via a *pull request*.

The BioJS registry includes the latest version of the code published in npm. Versioning is a fundamental part of npm, it uses the standard known as SemVer (`http://semver.org/`) to specify if changes are: *patches*, where bugs or changes that don't alter the functionality of the package are done; (ii) *minor* releases that include new features that are backwards compatible; or *major* releases, when the new features have incompatibilities with previous versions.

Every time a package is publish in npm it requires a new version number. In this way users of a package can specify which version to install in order to ensure the correct behaviour of their components.

### 3.1.2 Developed Components

Besides the contributions in the development and maintenance of BioJS we have created a few BioJS components that are currently listed in the registry. The components described below were originally developed for BioJS 1.0, but have been updated to version 2.0. This process was manually implemented in the case of the Chromosome Viewer, in order to explore and test the new features of the registry. In contrast, the upgrading of the PPI components, mentioned at the end of this section, was achieved with a tool developed to upgrade all the components from the first version into BioJS 2.0.

An article that is part of the BioJS series in the F1000 research journal describes the components for the display of PPI networks using a force directed layaout and a circle layout. The first author of the publication is Gustavo A. Salazar and the co-authors provided input in their role of collaborators and supervisors.

*Chromosome Viewer*

We designed and developed a BioJS component to display a chromosome that includes the reported cytogenetic bands. The original version of this component is available in the BioJS 1.0 registry (`http://www.ebi.ac.uk/Tools/biojs/registry/Biojs.Chromosome.html`) and there is also an updated version for BioJS 2.0 (`http://biojs.io/d/biojs-vis-chromosome`).

Cytogenetic bands are the result of an experimental technique that dyes chromosomes in such a way that regions with higher percentage of the nucleotides adenine and thiamine are dark, while guanine and

cytosine rich areas are light. This colour code is usually a good indication of areas with less (dark) or more (light) genes. The exposure of this bands help in the identification of chromosomes, and it is also useful to easily localise a genomic region in a chromosome.

The naming convention for the bands in a chromosome divides the bands in two groups physically marked by the position of the centromere, which is the region where a chromosome links with its pair. This division creates a short and a long arm, which are marked as p and q respectively. The bands are then numbered starting from the centromere as p1, p2, etc. for the short arm, and q1, q2, etc, for the long one; and more numbers can be used to name sub regions that are detected in higher resolution [78].

**Figure 3.1.4:** BioJS component to represent a Chromosome

Figure 3.1.4 is an snapshot of the BioJS component displaying chromosome 8 from *Homo sapiens*. The information is obtained from the DAS server provided by Ensembl that contains the karyotype information (`http://www.ensembl.org/das/Homo_sapiens.NCBI36.karyotype/features?segment=8`). The DAS response is processed using JsDAS, extracting information about the chromosomic location, name and category of each band, which ultimately defines where, what label and which colour will be used to paint the region in the component.

The chromosome is represented in a web page by a composition of HTML div elements, where all the divs are drawn on the same line without any separation between themselves. CSS classes are defined for each of the chromosome categories, for example the style below will be assign to the div that is rendered with a gpos25 category.

```
1 .gpos25 {
2   background-color: rgb(25%, 25%, 25%);
3 }
```

The border of the bands at both edges of the chromosome and the ones next to the centromere are rounded in order to visually identify these areas (as seen in figure 3.1.4).

The component broadcast events when the user hovers or clicks over a band, in the examples this is used to display a label or to reposition an area selector over the corresponding bar. This area selector is an utility BioJS component (`http://www.ebi.ac.uk/Tools/biojs/registry/Biojs.AreaSelector.html`), that although is not biological related, it can be reused by any component that uses a similar approach to the graphic representation (i.e. HTML elements). Figure 3.1.4 shows the selector over the band that is at the left of the chromosome. The selected region can be manipulated by dragging the edges of the blue area,

and every time a new area is marked, events are broadcasted and other BioJS component can react accordingly.

In BioJS 2.0 is recommended to only publish biological related components, therefore we decided to exclude the Area selector component from the BioJS registry. Nonetheless the component was published in npm (https://www.npmjs.com/package/area_selector) and can be reused by other npm packages including those part of BIoJS

*Protein-Protein Interactions Force Layout*

The most popular way to visualise networks is known as the node-link metaphor, in which some figures (e.g. circles. squares, images) represent the nodes; and lines connecting them are the links. When the network consists of a short amount of nodes, it is possible to specify the position for all of them, however when the amount of nodes grows, the effort required to position all of them also increases and random location of nodes would probably generates highly congested graphics with overlapping of elements and very cluttered areas.

Different strategies have been defined to automatically organise network representations. We have used the implementation of a force-directed layout included in the D3 library (https://github.com/mbostock/d3/wiki/Force-Layout) [9] for the BioJS component that can visualise protein-protein interaction networks in a web native component:
http://www.ebi.ac.uk/Tools/biojs/registry/Biojs.InteractionsD3.html,
http://biojs.io/d/biojs-vis-interactions-d3.

The force directed layout implemented in D3 follows the simile of attaching springs to all the connected nodes, in this way two nodes connected pull towards each other, and if a node has several links it is subjected to all the spring forces of its connections at the same time. The algorithm also includes a rejection force between all the nodes, which aims to disperse unconnected nodes and avoid cluttered areas. An optional gravity point can be setup where all the nodes are attracted, this helps to keep the nodes in the visible area.

The simulation of the movement of the nodes caused by the defined forces is executed by steps. For each step, all the forces affecting a node are used to calculate a new position for it, and this is done for all the nodes. An internal temperature parameter is used to control the speed of change during each step: when the simulation starts the temperature is high and the particles move fast, and with each iteration the network gets closer to an stable position dropping the temperature, and the nodes move slower as a consequence. The use of the temperature parameter also helps to stop completely the simulation when is lower than a predefined value, this with the purpose of avoiding unnecessary CPU use.The simulation might be required to restart when for example new nodes or links are added to the network, in which case

the temperature parameter is changed to a high value.

In order to optimise the number of operations required on each step, the D3 implementation includes a data structure called quadtree that creates a hierarchical organisation of the nodes based in their location. This is used when calculating the rejection forces of each node, so instead of considering all the rejection forces, an aggregated value for the nodes in a region can be used.

A quadtree divides the area in quadrants, the root of the tree contains all the nodes and it has four branches. The first branch contains the nodes that are located in the first quadrant of the graphic (i.e. between the top-left corner and the centre of the graphic), the second branch includes the nodes inside the second quadrant, etc. The same division can be recursively calculated for each quadrant.

The decision on how deep in the structure the algorithm should go to calculate the rejection forces is based on the criterium known as Barnes-Hut [6]. If the quotient between the width of the quadrant $s$ and the distance from the node to the centre of the quadrant $d$ is less than a threshold value $\theta$, the aggregated force of the quadrant is sufficient and the recursion on the tree can be stopped.

An step in the simulation is concluded when all the forces of all the nodes have been use to calculate the new positions of all the nodes. And every time an step concludes, an event called *tick* is broadcasted. Listeners of the event, can then alter the behaviour of the simulation by adding other forces, for instance to contain the nodes in a frame, or to define several gravity points. The event is also useful to trigger changes in the graphic representing the network, and in this way to animate the simulation.

For the BioJS PPI force layout component we used the *tick* event not only to display the simulation but also to define multiples *foci* of gravity, one for each organism reported as a protein, in this way the proteins that belong the same organism get attracted to the same area.

The representation of the network in the browser has been implemented using SVG. Path elements forming symbols are used to represent proteins of different organisms. We took advantage of the SVG technology by allowing the user to freely zoom and pan around the graphic with popular mouse gestures i.e. scrolling for zooming and dragging for panning, owever, if the user is dragging the mouse over one of the figures representing a protein, only that node gets moved and once the mouse id released, the protein is forced to stay in the position were the user drops it, this action restarts the simulation in order to find positions for the rest of the proteins in the graphic.

A running example of this component has been published in `http://jsfiddle.net/Bvh6k/8/` using JSFiddle (`http://jsfiddle.net/`), an online resource that supports the edition and live execution of snippets of Javascript code. Details on the biological entities used in the example can be found in [97]. Figure 3.1.5 is an snapshot of the visualisation generated by this example.

Besides allowing developers to experiment with the code, the example serve us to explain the use of the component.

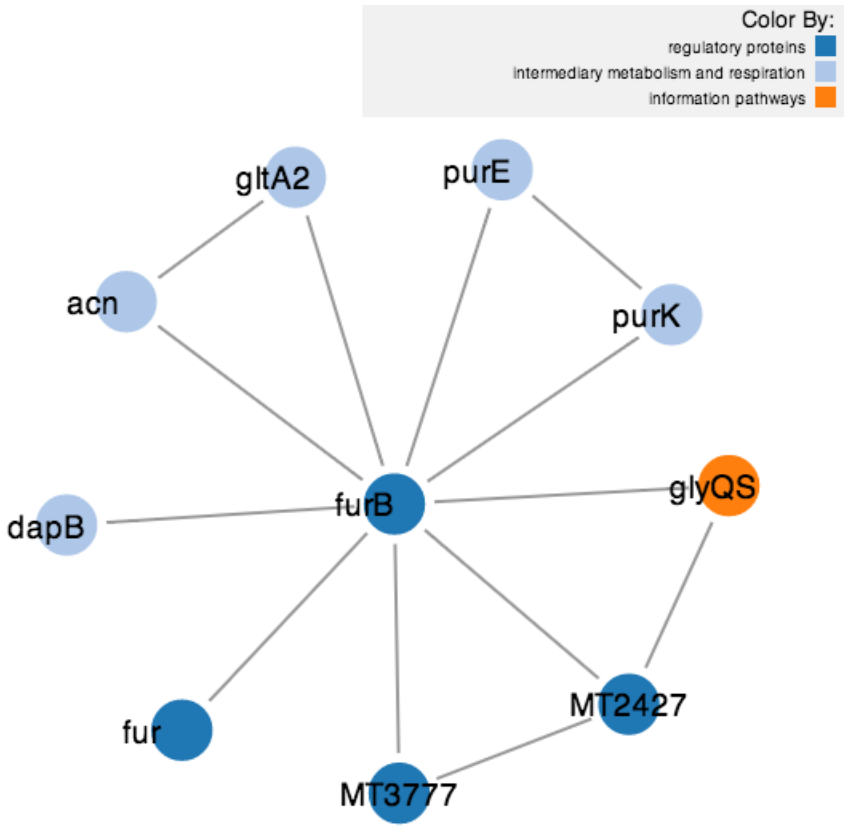


**Figure 3.1.5:** BioJS component to represent a network using the force-directed layout

A developer should start by creating an instance of the component:

```
var instance = new Biojs.InteractionsD3({
   target: "example"
});
```

All the proteins in the graphic should then be added. The following example shows how to add the protein with UniProt id O05839 (http://www.uniprot.org/uniprot/O05839). Note how the organism to which it belongs is included, along with which feature should be used for the label:

```
instance.addProtein({
      id: "O05839", organism: "MTB",
      gene_name:"furB",
      typeLegend:"gene_name"});
```

In the same way, the interactions should be added to the graphic, making sure that the interactions are between proteins that have already been added. For instance:

```
instance.addInteraction(
      "O05839", "P0A582",
      {id: "P0A582_P0A582"});
```

Once all the proteins and interactions have been declared, the graphic can be restarted so it reflects the additions:

```
instance.restart ();
```

The example also includes some instructions for colouring and format.

*Protein-Protein Interactions Circle Layout*

We have developed another component to display PPI networks that uses the D3 implementation of a circle layout: http://www.ebi.ac.uk/Tools/biojs/registry/Biojs.InteractionsBundleD3.html. In this layout the proteins are organised in a circle, positioning all the nodes around the centre without favouring any of them, this can help to discover connection patterns around the network (Figure 3.1.6).

The interactions in this visualisation are spline curves which path is defined through the hierarchical edge bundle algorithm [43]. This algorithm requires the data to be organised in a hierarchy, and in the case of been used in combination of a circle layout, the non-leave nodes of the tree are organised following a radial pattern, where the root of the tree is in the centre, and first level children are located in a circumference around it, the nodes of the following level of the tree are position in a circumference of bigger radius around the root. The same is then done for all the levels of the hierarchy. The nodes in the last level, which are the leaves of the tree, are the only visible ones.

When an interaction between two nodes is drawn, the curve's path will be created using the hierarchy nodes as guidelines. This method highlights the links between two highly connected groups in the hierarchy, because all of those connections will follow a similar inner path, creating the impression of a bundled connection, hence the name of the algorithm.

The hierarchy used for the BioJS component to display PPI network is a two level tree, that simply separates the proteins by organism, which is useful when looking to multi-organism networks such as a host-pathogen scenario. Another case of multi-organisms is discussed in section 4.1, where pseudo-interactions were defined to identify a relationship between two orthologs.

Figure 3.1.6 shows an snapshot of the component displaying the same dataset as the example for the force-layout component, which has been also uploaded as a JsFiddle: http://jsfiddle.net/J4CE7/9/.

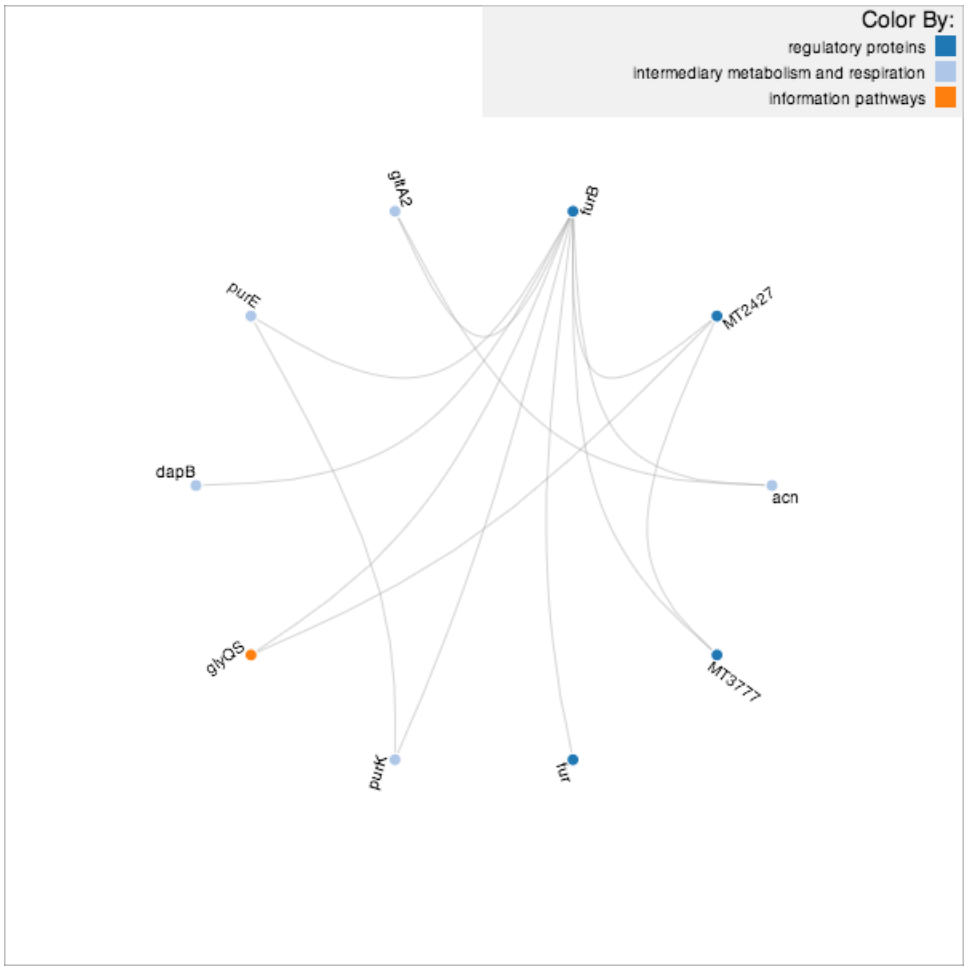


**Figure 3.1.6:** BioJS component to represent a network using the circle layout

Both components follow the same API (Application Program Interface) and therefore any script developed to display on one layout can be used on the other. The only difference between the APIs is because of methods that help to control the force layout of the first component which do not apply to the Circle layout. Thanks to this, the script to generate a PPI visualization of the same network is very similar in both. The only difference between the two scripts loaded in JsFiddle is the declaration of the object. Instead of using the object Biojs.InteractionsD3 it uses Biojs.InteractionsBundleD3 and rest of the script is exactly the same.

```
1 var instance = new Biojs.InteractionsBundleD3({
2     target: ”example”,
3 });
```

*Protein-Protein Interactions: Heat Map*

After the publication of [97] where we described the two components explained in the previous sections, we decided to explore an alternative to visualise PPI interactions. A new BioJS component was developed to visualise the interactions as a matrix of proteins by proteins, similar to the way that some heat maps are used in the processing of microarrays or phylogenetics analysis, but in this case not all the squares of the matrix are filled, only those marking an interaction between two proteins are painted. The component is now in the BioJS 2.0 registry (`http://biojs.io/d/biojs-vis-interactions-heatmap-d3`) and the source code is freely available in GitHub (`https://github.com/4ndr01d3/biojs-vis-interactions-heatmap-d3`).

This method to visualise PPI networks put the emphasis of the graphic in the interactions rather than in the proteins. And by sorting the matrix and using a correct colouring of the protein labels, for instance based in the functional class, it is possible to find features that other wise aren't evident in a node-link representation because the overlapping of the interaction lines.

Figure 3.1.7 is a snapshot of this component in which random interactions of 15 fake proteins are displayed. We have included extra functionality in the example to demonstrate the use of some of the methods and events provided in the API of the component. The mouse pointer in the figure is located over the cell that remarks the interaction between *prot_2* and *prot_15*, hence the lines crossing over it. We have also clicked on that interaction cell, which triggers an event that displays two panels with the feature information of the interacting proteins. The size of panels and graphic have been optimised for better use the available space on the SVG container.

The interactions in the example have been coloured using a scale between red and green based in the other this proteins are included into the model. This feature can be used to represent any quantifiable value associated with the interaction, for example evidence scores.

An interactive example of this component can be accessed directly form the BioJS registry (`http://workmen.biojs.net/jsbin/biojs-vis-interactions-heatmap-d3/simple_example`). In this case we are using a different tool called JSBin, which behaves similarly to JsFiddle, the tool used in the examples of the previous components. JSBin has a better support for the inclusion of npm packages and therefore it is easier to use with BioJS 2.0 components.

The component also shares the same API as the previous two PPI visualisation widgets, which makes it easy to alternate the view between the 3 PPI components and in this way offer to the user several options to display their data.

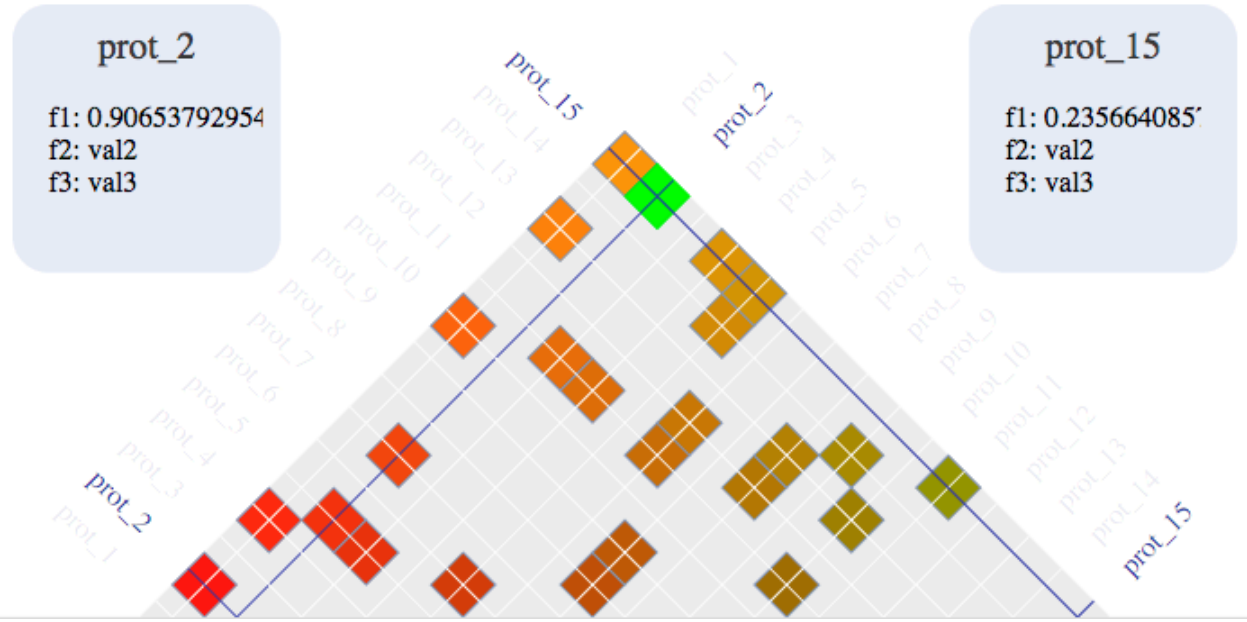


**Figure 3.1.7:** BioJS component to represent a network using a heat map

### 3.1.3 Discussion

We have discussed in the introduction (section 1.2.3) the existence of several tools that visualise PPI networks; all of which support variations of a force-directed layout. However by the time we developed the components described in this section, there were not web components to create such visualisations using HTML5 web standards. We are aware now of the existence of the Cytoscape.js project, which offers an alternative based in the HTML5 element canvas and include implementations of not only force-directed layouts but also circle, grid, and others. Nonetheless, we consider that is important to have technology alternatives for this type of components. If for example, a developer is already using D3 in its web tool, the use of our components can be a better alternative to reduce the network latency created by many dependencies.

## 3.2 PINV, a web-based Protein Interaction Network Visualiser

Protein-protein interaction data has been used in multiple research scenarios: (i) to browse networks for genes of interest, (ii) to interpret the results of genome-wide genomic screen, (iii) to interpret functional genomics data and (iv) to elucidate disease genes [29]. In all of them visualisation has play an important role, for instance, to meaningful navigate around a big network (i and iv), to find clusters of proteins with shared functionalities(iii and iv) or highlight links that are not evident with other techniques (i and ii).

Multiple projects to visualise PPI networks have been discussed in section 1.2.3, however none of them uses the web as their platform and in consequence their collaborative capabilities are limited.

The potential of community oriented projects built on the web has been exposed in other fields, for example GitHub has become the most popular system to collaborate in open source projects, with more than eight million users and over twenty million repositories by the beginning of 2015 (`https://github.com/about/press`), and although the core of it is the git version control software, it is arguably that the web front end is the one who attracts the community to take part of many open source projects.

We have developed PINV, with the goal of bringing this potential into the research of PPI networks. An article describing the project was published in 2014 and is listed as No. 2 at the beginning of this chapter. The first author of this publication is Gustavo A. Salazar and the co-authors provided input in line with their roles as collaborators and supervisors. Below we describe PINV in more detail, including its architecture and implementation.

### 3.2.1 Description of the application

PINV can be accessed in the web via this URL: (`http://biosual.cbio.uct.ac.za/pinv.html`). The visitor is presented a menu with four options: Data sets, Load data, Documentation and about.

The *Data sets* page is the default, here the user can find a list of all the public datasets that have been upload into PINV. We have loaded some public datasets for organisms of our particular interest: *Homo sapiens* (HS), *Mycobacterium leprae* strain TN (MLP), *Mycobacterium smegmatis* strain $MC^2155$, *Mycobacterium tuberculosis* strain CDC1551 (MTB) and a combined dataset of *HS*, *MTB* and the interactions between them. The rest of the datasets on the list have been uploaded by PINV's users. We have included a search functionality in the page where users can include part of the name of the dataset and the list is automatically updated to only include those that match the given text.

The second option in the home page is *Load data*, here a web form is displayed. The required information consist in a unique name for the dataset, a valid email where the user will receive the links to access the dataset, the preference between a public or a protected dataset (difference explained in the

next section), and two tab separated files: one for interactions and one for features.

In the interaction file, the first two columns contain the accession numbers of the proteins, and the third column is an aggregate score of the evidence for the reported interaction. Subsequent columns are optional and considered as evidence scores that contributed to the unified value in column 3. The second file is used to add annotations to the proteins (e.g., gene name). The only required annotation is that of the organism to which the protein belongs; therefore the file should have the protein id in the first column, the organism in the second and any other features in additional columns. The information provided in this file can then be used to manipulate the graphic, for instance to color by functional class or to show a particular label.

The last two options of PINV are *Documentation* and *About* where resources to provide help to the user are included. For example, tutorials, presentations and links to the source code and the original publication.

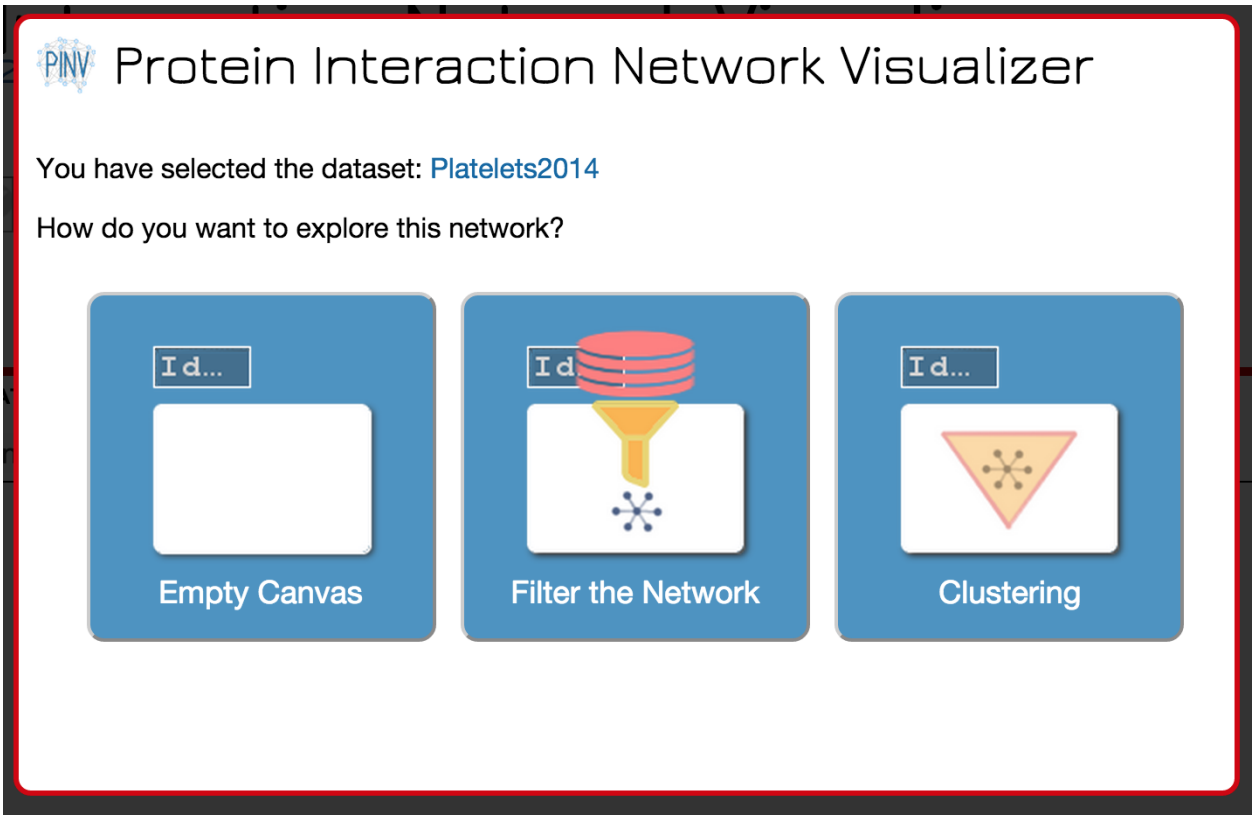


**Figure 3.2.1:** Wizard menu to select the exploration method in PINV

Once users select a data set from the list or access one from the link sent to their emails, the PINV client is load into the browser (See the client part in section 3.2.2) and then, a popup window(Figure

3.2.1) presenting three options to explore the data is displayed:

**Empty Canvas** Using this method the interface displays all the widgets, but no queries had been executed, and therefore the visualisations are empty. The proteins of interest can be selected by inserting comma separated accession numbers. The form has auto-complete capabilities to assist in this process (Figure 3.2.3-A).

The search function has three modes: Normal, Explicit and Recursive. The *Normal* mode returns the queried proteins and all interactions reported in the dataset, while the *Explicit* mode will only display proteins that have been explicitly included in the query list and the interactions between them; the explicit mode will also include all the interactions with the previous results. The *Recursive* mode combines the previous two, it first gets the target proteins on *Normal* mode, and then for each recovered protein it triggers a query in *Explicit* mode, the outcome of this is that all the interactions between displayed proteins will be shown.

**Filter the Network** In order to meaningfully visualize protein interactions, it is beneficial to reduce the network to only those proteins of interest through the application of prefilters to the data. This has the simultaneous benefits of increasing the speed of PINV's response time and reducing bandwidth requirements. By using the prefilter tool on PINV, the user can explore the whole content of the network without loading the individual components into the graphic. Figure 3.2.2 (Left) shows the interface where the user can define certain conditions that the interactions have to fulfil in order to be displayed. The graphic on the left side of Figure 3.2.2 is dynamically updated, reflecting the size of the subset generated by applying the filters over the whole dataset. For instance, in the graphic the dark blue represents the whole network (796353 interactions), while the light blue is the portion of those that have an evidence score from STRING [29]. Two more filters were applied to this example (*organism contains tuberculosis* and *description contains Sensor*), the combined use of the filters results in a 103 interaction subnetwork, which is small enough to allow PINV to run smoothly and generates a graphic like that in Figure 3.2.2 (Right).

**Clustering** In this mode the whole network is represented by clusters of proteins and interactions between them, the user can then expand the clusters, to explore the network in a top-bottom fashion. A more detailed description of this method is included in section 3.2.4.

PINV provides four types of visualisation of the data: Network, Circle, Heatmap and Table. The first three types were implemented as BioJS widgets and have been described in section 3.1.2.

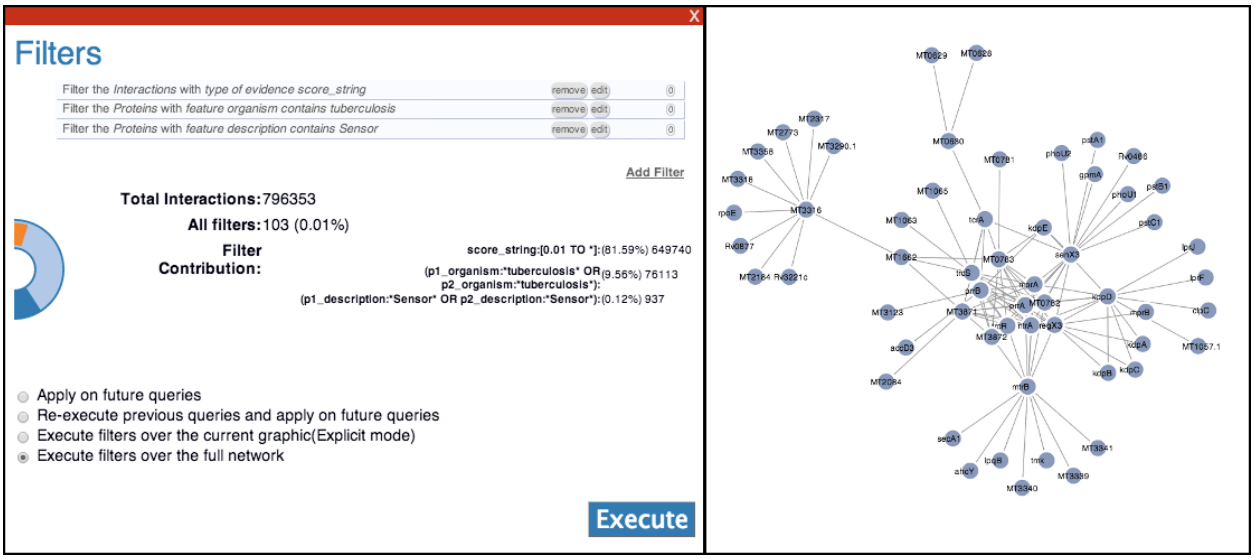


**Figure 3.2.2:** PINV Filters Dialog. Snapshot of the dialog where the user can explore a dataset by defining filters.

Figure 3.2.3-B displays a snapshot of the interactions for proteins O53166 and O05839 from the MTB dataset in the Network layout. The network layout allows the re-organization of the proteins through drag and drop mouse gestures, and clicking on a node will display the loaded features that are hyperlinked to relevant public resources in a pop-up window (right side of Figure 3.2.3-B).

The second layout organises the proteins in a circle positioning all the nodes around the center without favouring any of them, this can help to discover connection patterns around the network (Figure 3.2.3-C). In this view the interactions of a single protein are highlighted when the user hovers over the node. As in the network layout, the proteins are separated to clearly indicate the organism to which they belong; the user can choose to sort the proteins around the circle by any of the protein features.

Figure 3.2.4 displays the same query of figure 3.2.3 using the heat map component. In this representation the focus is in the interactions, and the proteins are only labels of them. This visualisation offers the possibility to compare the features of the two proteins involved in a interaction.

Lastly there is the table view where the raw data is displayed and can be reorganised. This is particularly useful for queries that return too many results and require post-filtering to select the proteins of interest.

The visualisation of the layout can be exported at any time into SVG or PNG format, and the data in the table are exportable to a CSV file.

Manipulation of any of the three display options is achieved with a single component (Figure 3.2.3-D) that allows the creation of rules. The rationale behind this component is that by specifying a target (protein or interactions), a condition to filter by (e.g., protein with functional class X, interactions with protein Y), and an action (e.g., paint green, show label) it is possible to manipulate a wider set of network

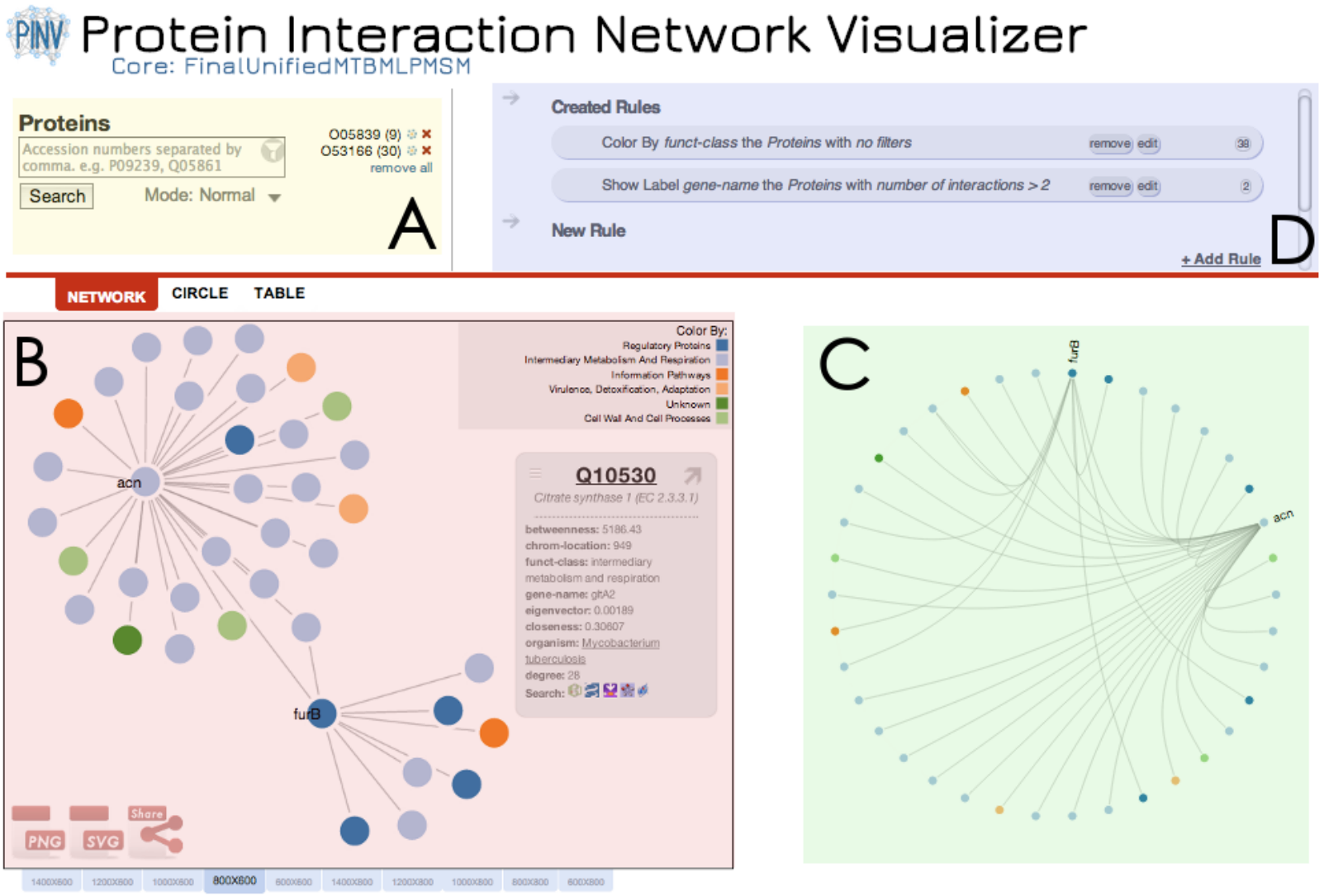


**Figure 3.2.3:** PINV Snapshot. (A) Search Form. (B) Network layout. (C) Circle layout (D) Manipulation by Rules.

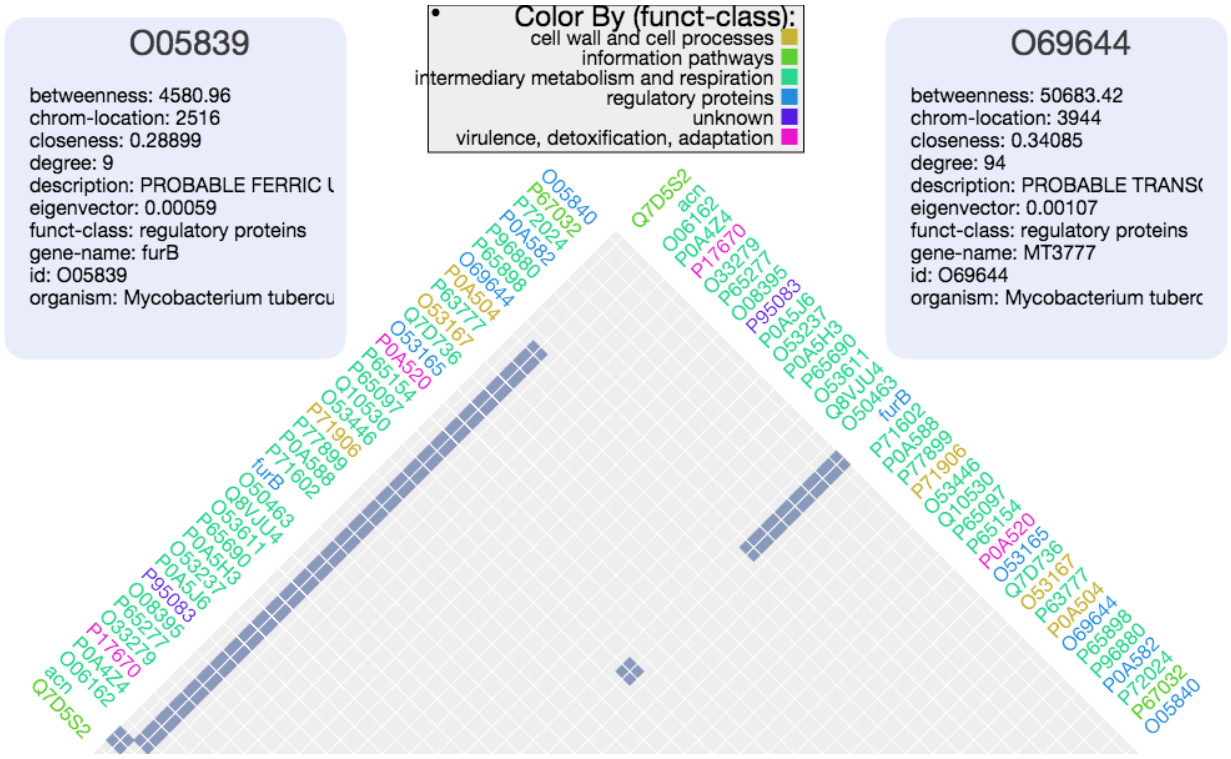


**Figure 3.2.4:** PINV Heatmap Visualization of the proteins O53166 and O05839 from the MTB dataset.

elements with less input from the user. For example, Figures 3.2.3-B and C show all the proteins painted in different colours depending on their functional class. The legend on the right lists the functional classes used and the corresponding colours in the graphic. A second rule was created in the example to display the gene name at the side of proteins with more than 2 interactions. Right clicking on a protein in the graphic will open a contextual menu that allows the quick creation of rules that only apply to the selected protein. Users can also colour nodes with quantitative data such as expression values loaded from a third file.

All the actions executed by the user have been saved in order to display the provenance of the visualisation in the form of a list where the user and collaborators can review the steps (and their parameters) taken to reach a particular status of the visualisation. This list can be displayed by clicking on the "Show History" button at the bottom of the page.

PINV provides an easy way to share the current view: when the user clicks on 'share', a unique link is generated. PINV is able to regenerate the status of all the components to re-compose the view whenever the link is used. This functionality was developed with the goal of promoting collaboration between users. If for instance, a user finds something of interest in the visualisation, he/she can create the link and send it to a collaborator via mail, instant messaging, etc. The second researcher can not only view but

manipulate the visualisation and re-create another link for future discussions. For example, recreations of Figures 3.2.2 and 3.2.3 can be explored in PINV by following the URLs
http://biosual.cbio.uct.ac.za/pinViewer.html?status=paperF and
http://biosual.cbio.uct.ac.za/pinViewer.html?status=paper00 respectively.

The sharing option also provides an HTML snippet that can be used embed the view in a web page, making it easier for the researcher to include their visualisations on web documents in the same way that maps, videos and other kinds of multimedia are included in blog posts. It is hoped that researchers will use this to embed PINV graphics in pages that discuss the results of their own network research.

### 3.2.2 Architecture

PINV has been designed to follow a classic client-server architecture, in which the client is in charge of the interaction with the user and the generation of the visualisations, based on asynchronous calls to recover data on demand. The server has been optimised to provide rapid responses through a REST interface, whilst the client follows a modular architecture that makes it easy to extend and adapt.

#### *Server*

The server side of PINV has been developed using the web framework for Python called CherryPy (http://www.cherrypy.org/) and the PPI data is indexed and stored in a Solr search engine server [60] in order to perform rapid queries, resulting in quicker response times than through querying of a relational database

PINV's client never interacts directly with the Solr instance, instead all communications are tunnelled by the CherryPy component, this with the purpose of prevent the access to resources and functionalities to unauthorised users. For example, the REST services of Solr includes a method to delete a core, which is the term used in Solr to describe the dataset and its controllers. We filtered out such calls and any other administrative functionalities that can harm the data of other users.

The main functionality of the server is to allow the upload of the PPI data from simple tab separated files into a Solr core. When uploading data, the user can choose between two modes of privacy: public and protected. A dataset in public mode will be listed on PINV's website and accessible to anyone. This brings visibility to the data and promotes collaboration between researchers. On the other hand, protected mode ensures that only users with a valid link can retrieve information from the data set; the link includes a unique key that is sent to the uploader so he can distribute the link as he wishes. In this way the collaborative features of PINV are still functional between the holders of the link.

The authorisation to a protected core is another function managed by the CherryPy server. The access to any information of a protected core is restricted to the inclusion of the key into the request. When data

is upload to PINV in protected mode, an email including links with the key in it will be sent to the user. A similar email is also sent to owners of public datasets, which include links to restricted administrative functions such as deleting the core.

Another feature carried out by PINV's implementation of the CherryPy server is to allow the storage and posterior retrieval of the files that contain the status the different widgets, which is the key element in the generation of links to share or embed a visualisation created in the tool. When the user clicks on the share button, the client compile the state of all its widgets in a single configuration file, including the current dataset, queries executed, graphic manipulation rules, etc.This file is sent to the server, where is stored and a unique identifier is assigned to it, the ID is then returned to the client which uses it to generate the URL and the HTML code to embed PINV in another page. When the link is used, the PINV client loads normally (See following section), but in addition the status email is recovered from the server and used to reload all the configurations and regenerate the visualisation.

Following the principle of making PINV an open source project, we have published the repository in GitHub (`https://github.com/4ndr01d3/pinv-server`) that includes the CherryPy logic, a Solr installation that has been configured to be used in PINV, an example configuration to use apache to redirect any Solr query into CherryPY, and documentation or all these components.

*Client*

When the user selects a dataset from PINV's home page, the main application gets loaded into the client, which initially means that two resources get retrieved from the server: a bundled javascript that contains PINV's scripts and all the widget files, and a similarly bundled CSS style file. These files are built on demand in the server by a PHP script that checks the configurations included in JSON file that is store in the server.

This schema allows modularity in PINV so when new widgets are developed, the core of the application remains the same and only the configuration files needs to be modified. It is also possible to have several configuration files to create instances of PINV with different widgets, for example the home page of the project executes the same routines, but includes different widgets to display the list of available datasets, the forms to upload data and the documentation of the system.

Figure 3.2.5 shows all the components that provide the whole client logic of PINV. As mentioned above, these components are compiled in a singled javascript file by a PHP script. Having a single file reduces the time of loading because it requires only one HTTP transaction.

**Main Dependencies** There are three main dependencies included in PINV: jQuery, AjaxSolr and BioJS.

PINV uses jQuery (`https://jquery.com/`) in several ways, for instance to manipulate data object

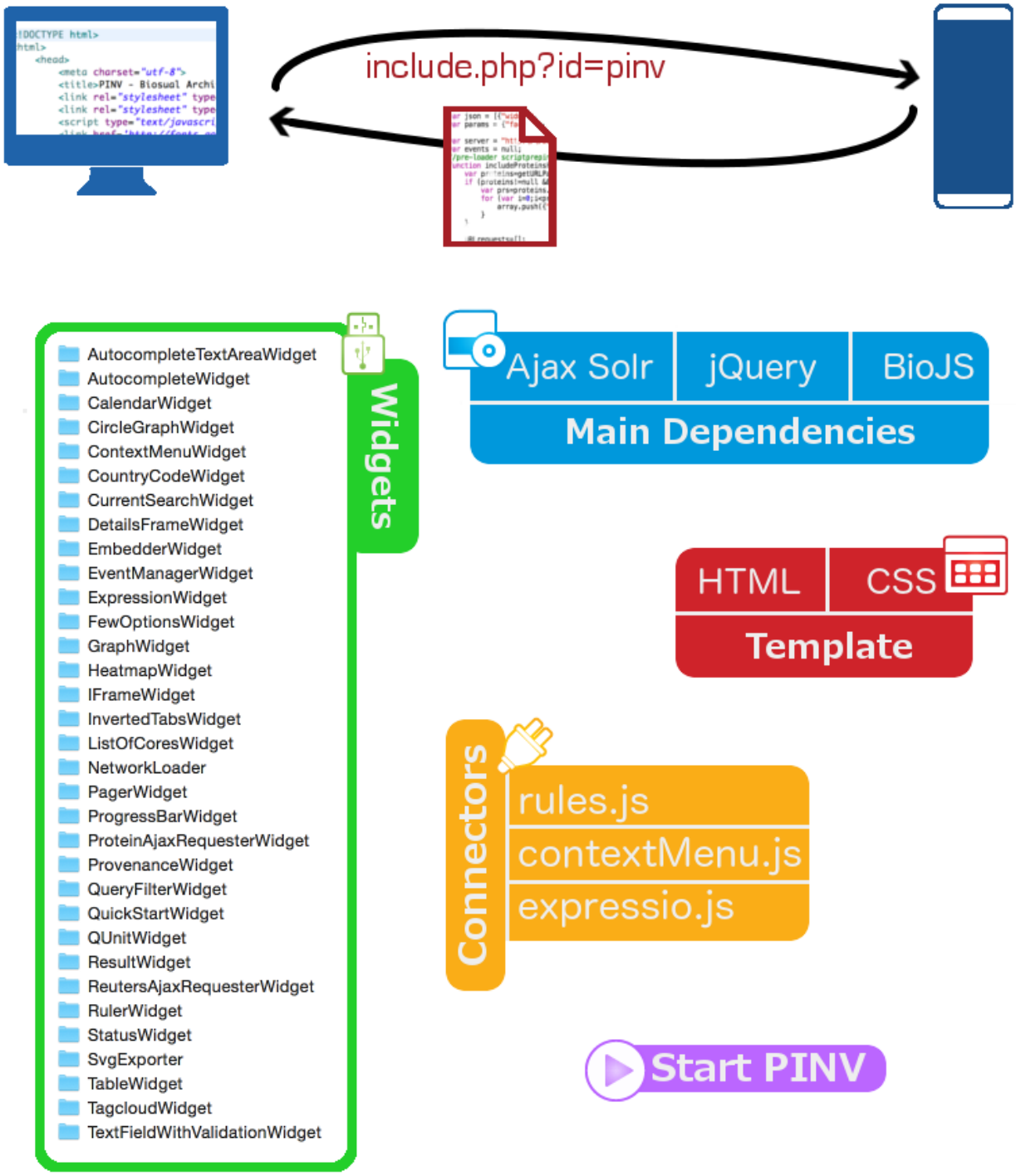


**Figure 3.2.5:** Files to be included by the PHP script of PINV

model DOM, which is the data structure that contains the HTML elements in web browser; this can be done without jQuery, but its syntax facilitates the process; jQuery is also extensible via plugins and PINV uses several of this extensions (e.g. draggable components, auto complete features, etc.).

The second general dependency is AjaxSolr (`https://github.com/evolvingweb/ajax-solr`) a library that serves as the interface with the Solr server that contains the PPI network data. AjaxSolr architecture follows a Model-View-Controller MVC pattern that has been adapted to the interaction with a Solr server: the model is based in the structure of the documents retrieved from the server; widgets are registered to the AjaxSolr core, which acts as the controller, notifying the widgets about the actions and results of interacting with the server. Widgets are in charge of generating the views, because they can include methods to react according to each notification, for example to display the results of a query once the core triggers an *afterRequest* event.

PINV reuses this strategy, and each of the components that form this application have been developed as AjaxSolr widgets, so they can get notified of each interaction with the server, some of the widgets use the core to execute new requests, while other wait for responses to generate views.

The main class of BioJS 1.0 is the last general dependency included in the compiled file. The visualisation widgets included in PINV have been implemented following the BioJS standards. Each of the components described in section 3.1.2 are used in PINV: the logic related with PINV was written as an AjaxSolr widget, but all the visualisation features were kept as a separated BioJS component. All of these components inherit from the BioJS class added in this step.

**HTML Template** After the dependencies have been included, the script proceeds to extract the information about the HTML template from the configuration file. The template is a HTML piece of code that works as layout for the components, specifying divs or other container elements where the widgets can be included in a later stage. In this step the PHP script creates a variable in the javascript code containing the selected template, and then a piece of executable is added to inject the template into the DOM of web page.

**Widgets** As mentioned before a widget in PINV follows the schema defined in AjaxSolr, however we extended this concept in order to include other resources related with each widget. In the server each widget has its own folder, which should contain a configuration file called widget.json. The PHP reads this file in order to extract the paths of each of these resources: JavaScript dependencies, CSS dependencies, HTML, CSS and the JavaScript widget. Only the JavaScript file is required, all the other resources are optional.

The code below is the configuration file used to describe the widget that displays the network using the force-directed layout using the BioJS component described in section 3.1.2. The *GraphWidget.js* file defines the widget as a class that inherits from *AjaxSolr.AbstractWidget* and implements the methods to initialise the widget and process all the interactions with the server; The widgets uses the BioJS component *Biojs.InteractionsD3.js* to visualise the interactions, which in times uses the implementation of the force-directed layout included in the D3 library (*d3.min.js*). The HTML in *GraphWidget.html* is limited to the definition of a div element where the component will be included.

```
{
    "name": "Interaction network",
    "description": "A network graph using D3",
    "dependencies":["d3.min.js","Biojs.InteractionsD3.js"],
    "js": "GraphWidget.js",
    "html": "GraphWidget.html",
    "css": "GraphWidget.css"
}
```

The current version of PINV includes widgets that facilitate the search in the server by ID(*AutocompleteTextAreaWidget*), to filter the query using the protein features (*QueryFilterWidget*), to visualise the interactions using a force-directed layout (*GraphWidget*), a circle layout (*CircleGraphWidget*) and a heat map graphic(*HeatmapWidget*). This visualisations can be manipulated by other widgets, for instance by creating rules to define colour or size depending on the protein features (*RulerWidget*), or by using expression data to paint the proteins on a corresponding colour scale (*ExpressionWidget*). There are other widgets that were created to provide some GUI features, such as the tabs to select the visualisation method (*InvertedTabsWidget*) or a floating window that can display the features of a selected protein or interaction (*DetailsFrameWidget*).

All the widgets were developed to be able to be executed independently. This means that the whole architecture can be used for a different application, and if the widget applies then it can be used there, for instance the autocomplete text area widget is not attached to protein protein interactions data, and therefore it can be used to generate queries by ID in any other configuration of the system.

**Widget Connectors** In order to keep the widgets reusable, we have separated the logic of their interactions in separate files. The widgets implement the triggering using the common event handler included in BioJS 1.0, however the listeners of the event are not declared in the widget

itself, but in the connector files. In this way two configurations of the system can operate differently based on the same event, but still using the same widget.

Consider for instance, two configurations of PINV: in the current version there is a tab widget that allows to select which visualisation method is going to be used; alternatively, we can have a configuration were all the visualisation methods are displayed simultaneously. In the first scenario there is not need of communication between the visualisation widgets, while in the second, it is desirable that a click in the symbol that represents a protein in one widget, triggers a reaction in the correspondent elements of the other widgets.

PINV architecture support this two behaviours without requiring changes in the files that define a widget, but instead including this type logic in the *Widget Connectors* files. The three files that the current version includes are supporting the events generated by: the rule definer widget (*ruleExecuter.js*), the context menu that get displayed when a right click is done over a protein (*contextMenuFunctions.js*) and the widget that processes the expression files (*expressionExecuter.js*).

These files are only necessary when the interconnection between widgets requires some logic, for example to filter the visible components based on the defined rule, or to interpolate an RGB value based on the limits and colours selected by the user. For other events where the interaction between triggered and listener is more direct, can be simply reported in the configuration file, for example when a click in protein X in a widget, triggers the method (highlightProtein()) in another widget.

**Starting logic** All the files included until this moment are either class definitions or injections of HTML to build the layout of the page, it is only at this stage of the file where the routines to execute all this components are added.

As explained before, the logic of operation follows the same strategy than AjaxSolr, and therefore the main part of this logic is to instantiate this class and to register all the widgets that are being used.

The configuration file supports the inclusion of a preloaded file, in case there is some logic out of the widgets that needs to be executed previous to the initialisation of widgets. Currently this is used to retrieve the schema of the dataset that a user has selected because each dataset can have different protein features, and this information is used for some widgets, for example the ruler needs to know this feature names in order to build the forms where the user build the rules.

Similarly there is the possibility of including a post loader script, however the current configuration of PINV doesn't make use of this option.

### 3.2.3 Comparison with Similar Tools

Most interactive online visualization tools only allow the viewing of networks that are preloaded, meaning that the user cannot view a network he/she has generated.

The online version of Graphle [47] is limited to graphs generated by bioPIXIE in yeast, MEFIT in *E. coli*, or HEFalMp for human data. STRING [29], a database for known and predicted protein-protein interactions, and STITCH [61], its sister project for chemical-protein interactions, integrate interaction information from various sources in addition to in-house predictions. Both projects use the same library to display their data. However, the network visualized in both databases is barely customizable. The public implementation of VisANT [44], a web-based (via Java Applets) workbench for the integrative analysis of biological networks, is based on the Predictome database. PINV on the other hand, allows the user to view preloaded data but also to load their own data.

Another alternative is Cytoscape Web [67]. However, the fact that its introduction tutorial (`http://cytoscapeweb.cytoscape.org/tutorial`) requires its users to code in JavaScript indicates that this tool is mainly intended for developers to display networks on the web, not for the scientist who has a network to visualize. In this regard, Cytoscape Web was considered as an alternative to D3 for our background technology to generate the graphics as it is closer to the biological concepts tied to this tool. Unfortunately Cytoscape Web uses Adobe Flash for the generation of the graphics, which goes against our objective of providing a native web application (i.e. developed using recent web standards).

We also compared PINV with the stand-alone version of Cytoscape by using the network in the biological example mentioned in the following section. The original network contains all interactions from the three organisms and the orthologs with a total of 165,000 interactions. The performance of Cytoscape notably exceeded PINV when dealing with a network of this size. Moreover, there are considerably more manipulation options in Cytoscape than the ones currently provided by PINV. However, Cytoscape and its many plugins need to be installed by the user and any collaborator sharing the results.

We are aware that PINV will also struggle to display the 165,000 interactions of the network. However, the strategy of only visualizing by request and the use of prefilters allows the user to navigate the network by limiting the graphic to the interactions of interest. In contrast Cytoscape web does not provide tools to explore a large dataset and requires the use of the stand-alone application (or other software) in order to filter and create a subnetwork.

### 3.2.4 Spatial Clustering

A poster presenting the first version of PINV was exposed at the South African conference in bioinformatics in September 2012, which we consider as the release date of the tool. Many tools have

been included since those early days: the pre-filters, the different modes to query, the heat map view, the sharing capabilities and many small improvements and bug fixings, which we hope have contributed in creating a better software product.

A constant challenge during this time has been the scalability of the visualisations, which is not a particular problem of PINV but is common among any tools for network visualisations. This, however is stressed more in PINV because it is a web tool, and the resources available for this visualisation are limited to the ones available in the browser where is been displayed. Moreover, the dynamic representation and the direct manipulation options offered in the tool required dedicated resources for each element in the graphic.

Given the limitations imposed by the underlying technology, the larger the network to be displayed, the slower the performance of PINV. During an in-house test, PINV performed well with subnetworks of up to a thousand interactions, and thereafter performance decreases proportionally to the number of added proteins/interactions. Note that this is highly dependent on the specifications of the users computer.

In order to overcome this difficulties we have developed an algorithm for clustering that uses the positions that a force-directed layout provides. This with the purpose of offering if not a complete view of a network, at least an approximation of it.

*High level algorithm*

The idea behind the algorithm is simple: highly connected nodes get attracted to each other when imposing a force-directed layout, simultaneously, the rejection forces between nodes, makes that not so connected groups get spread over the graphic area. This creates visual clusters in the graphic. The idea is that instead of creating all the graphic elements representing the nodes and connection of the network, we only display cluster elements that symbolise the group.

The following pseudocode shows the logic of the algorithm. Given a network represented by a two arrays: nodes and links, we apply the force directed layout algorithm to obtain coordinate positions for each element, then we generate cluster elements based on those coordinates (a discussion to the creation of such clusters can be found later in this section). Inner connections are ignored (line 9) because they won't be displayed and interactions with elements out of the cluster are group; and outer connection will be group.

When painting the network, we can make the size of each cluster to be proportional to the number of nodes that is representing and the thickness of the line between two clusters to represent the number of connections in between them.

```
links = [...] //Array of the links in between nodes
force_directed_layout(nodes,links)
clusters = create_clusters(nodes)
cluster_links=[]
foreach link in links {
  cluster_source  = get_cluster(link.source,clusters)
  cluster_target  = get_cluster(link.target,clusters)
  if (cluster_source != cluster_target){
    //try to get an existing link between the clusters
    cluster_link = getLink(cluster_links,cluster_source,cluster_target)
    if (cluster_link == null) {
      // if there is not link, create one
      cluster_links.append( {
        source:cluster_source,
        target:cluster_target,
        number_of_links:1
      })
    } else {
      // if there is an existing link, increment the number of links between the clusters
      cluster_link.number_of_links = cluster_link.number_of_links +1
    }
  }
}
paint_network(clusters,cluster_links)
```

Figure 3.2.6 shows an example of executing the algorithm. The left side (a) is a node link representation organised through a force-directed layout, the network uses a dataset that reports the co-ocurrence of characters in the novel *Les Miserables*; we have chosen this dataset because it has been used in several examples for network representation in different software tools such as Cytoscape and the D3 library. Figure 3.2.6 (b) shows nodes in beige colour representing the created clusters. A interactive execution of this example can be seen in this URL: http://bl.ocks.org/4ndr01d3/2686646672a92aa89a0e.

Although the clusters are not perfectly identifying the predefined groups in the dataset, the overall shape of the network is conserved and it is easy to identify dense areas and highly connected clusters. Future work can be done in representing the cluster in a way where they represent its content, for example creating pie charts to show the contributions of different groups.

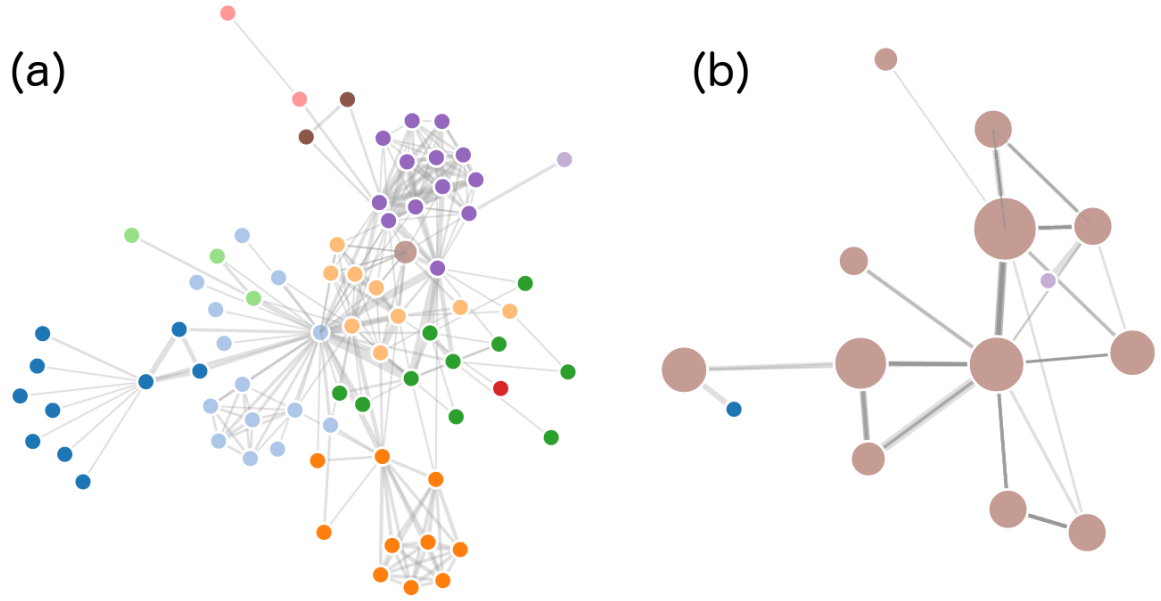


**Figure 3.2.6:** Spatial Clustering Example. (a) A networks organised by force-directed layout. (b) A cluster representation of network (a)

*Clustering with quadtrees*

The creation of the clusters can be achieved by executing any geometric clustering algorithm, such as k-means. The only restriction is it should use the geometric distance using the coordinates obtained with the force-directed layout, because in this case we are interested in clustering by the position in the graphic and not based in the similarities of the node's features.

However, we decided to create the clusters in an alternative way, inspired in the optimisation of the force-directed layout implemented in the D3 library. Which as discussed in section 3.1.2, uses an structure called quadtree to represent and group the nodes based in its location in the graphic.

Figure 3.2.7 serve us to explain the construction of a quadtree. In the left side there is an area where points have been randomly located, the algorithm divides the area in four equally sized quadrants, such as the area in light blue representing the bottom-right quadrant. A data structure is created to include the points covered in each of the quadrants. If a quadrant has more than one point, the logic is repeated recursively, saving the results in the same data structure, which creates a tree where all the leaves contain a single point.

The tree in the right side of figure 3.2.7 is the quadtree of the points in the left side. For instance the two points with red border are part of the bottom right quadtree(light blue area) which correspond to the four branch from the root of the tree in the left. Posterior partition shows the nodes in the sub-division

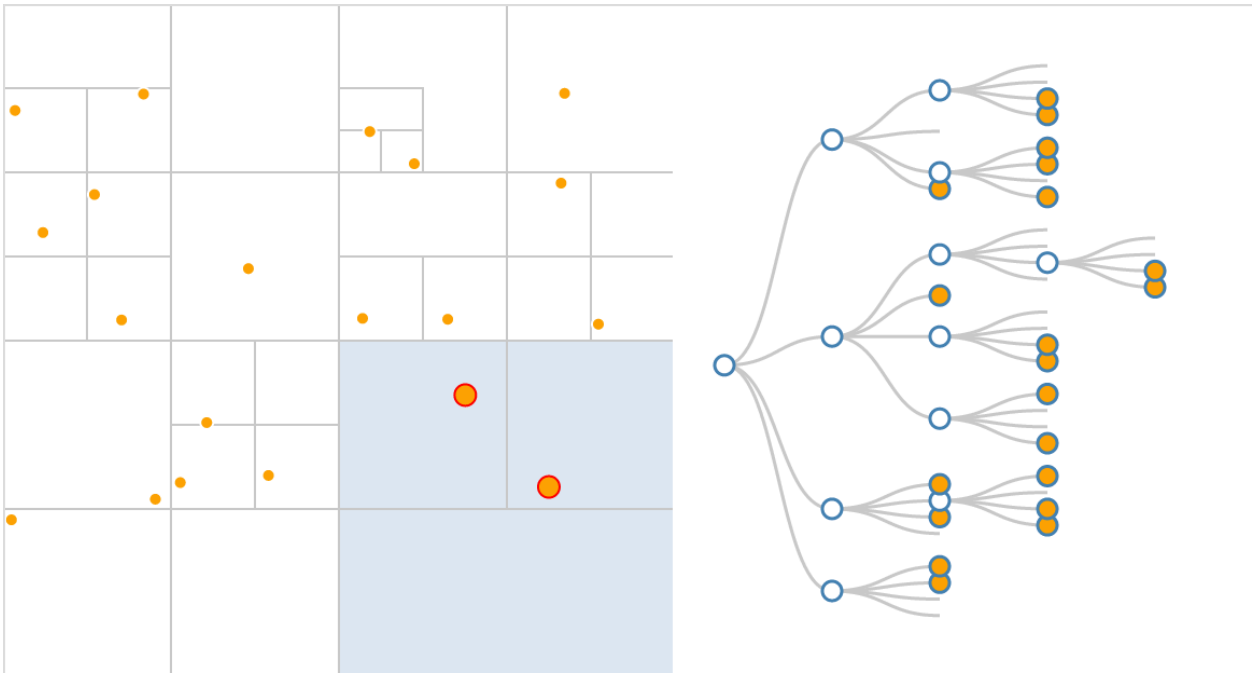

**Figure 3.2.7:** Quadtree representation of nodes in graphic.

top-left and top-right of the graphic, represented in branches 1 and 2 of the subtree.

This data structure has been used to optimise multiple area related algorithms, for example, to detect the collision between bodies: if a body is exclusively located in one quadrant, there is no need to verify for collisions with bodies in other quadrants, this is applied recursively through sub-quadrants, reducing the number of comparisons.

An animation that constructs the tree dynamically can be seen in this URL: `http://bl.ocks.org/4ndr01d3/727175afbdc58c3626b8`.

We have used the quadtree structure to define groups of nodes, and provide a way to explore a complex network on a top-bottom approach. The highest level of such representation contains four clusters, one for each quadrant, and the position of a representative element will be calculated as the average of the node positions in that quadrant. The user of such visualisation can choose the level of detail of the graphic representation, which gets interpreted as how deep in the quadtree should the algorithm go.

*Spatial Clustering in PINV*

We have implemented this algorithm in PINV in two different ways: one to be executed with the currently displayed graphic and one to represent a whole dataset.

**On the current graphic** Executing this algorithm with the current graphic is a task that can be

completely achieved in the client side. We used the coordinates of the nodes in the graphic, which have been already calculated with the GraphNetwork widget in PINV that uses the D3 implementation of the force-directed layout.

We have added a button with the label "Cluster" above the graphic that once activated runs the algorithm as shown in figure 3.2.8, clusters are drawn with grey triangles with red border and its size is proportional to the number of elements that they are representing. When the clustering mode is active an slider is visible at the side of the "Cluster" button, allowing the user to select the strength of the clustering, which in other words selects the exploring depth in the quadtree.

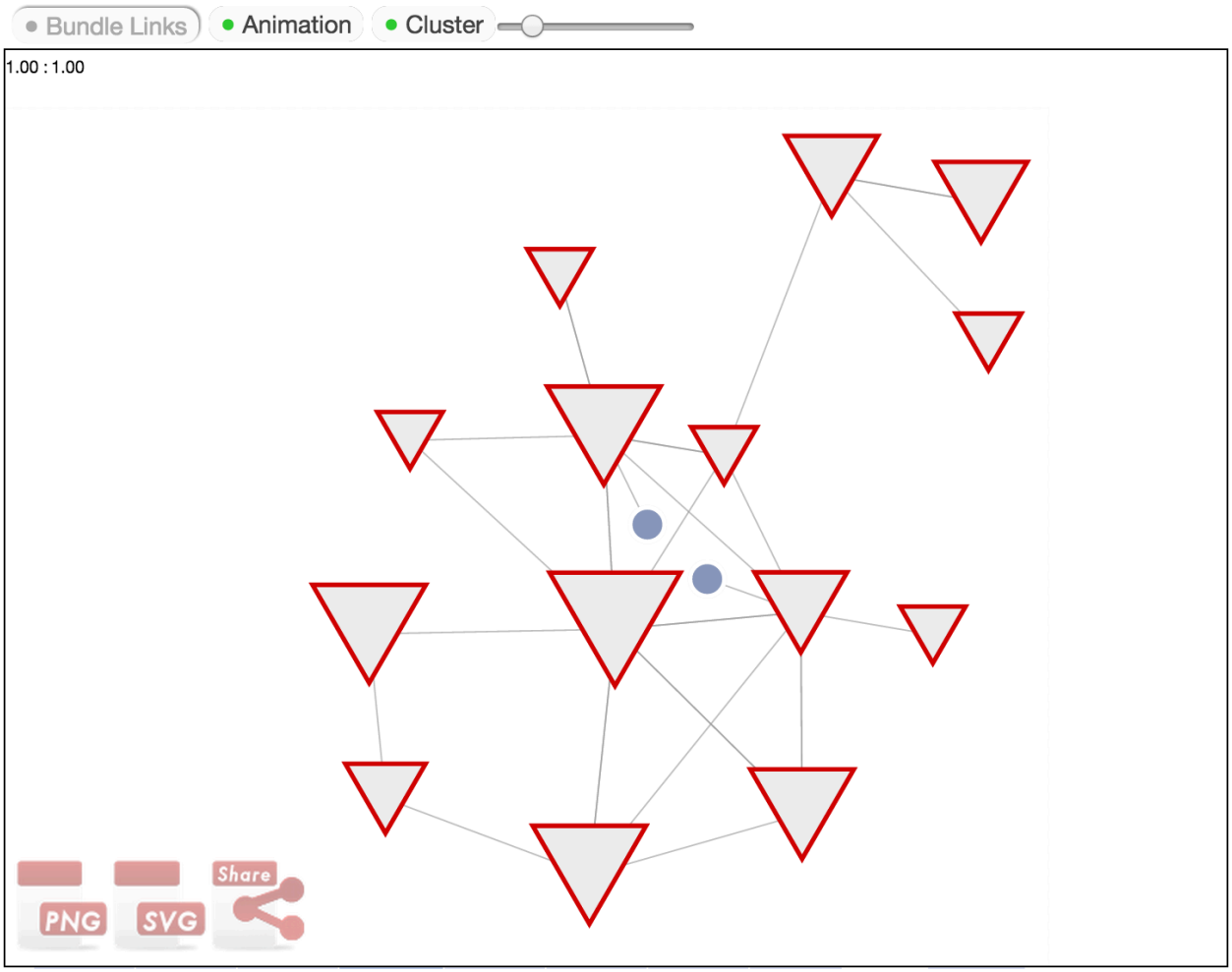


**Figure 3.2.8:** Clustering the current graphic. Triangles are elements representing a group of nodes, where its size is proportional to the number of elements.

If a leaf is reached in the tree before reaching the desired depth, it will be represented as a protein and not a cluster. Which is why the figure 3.2.8 has some circles interacting with a cluster.

**On the whole dataset** To be able to calculate a quadtree we required the positions of all the nodes to include. Therefore, either the PINV client requests the whole dataset from the server or the quadtree is calculated in the server. With the first option we could reused the implementation above, but the time required to do all the HTTP transactions to download all the interactions is a problem, however the bigger issue is the performance limitations of the browser.

Therefore we decided to generate the quarter in the server, which implied to implementf both force-directed layout and quadtrees into Python, the language used in our server. The implementation was simply a migration of the JavaScript code in the D3 library, with minor changes to adapted in to Python.

Now every time a new dataset get uploaded in to the PINV server, we execute the force-directed simulation, to obtain coordinates of an imaginary area that can contain all the proteins, using those coordinates we generate a quadtree, and save the interactions between clusters of the first seven levels.

When a user selects a dataset to explore in PINV a menu (Figure 3.2.1) to select the exploration mode is displayed as discussed in section 3.2.1. If the user chooses the third option *Clustering* PINV queries the server to recover the clusters of level 3, and therefore the graphic will contain up to 64 clusters, which is a number of elements that can be easily handled by the client, but more importantly it is a number graphic that it is more understandable by the user.

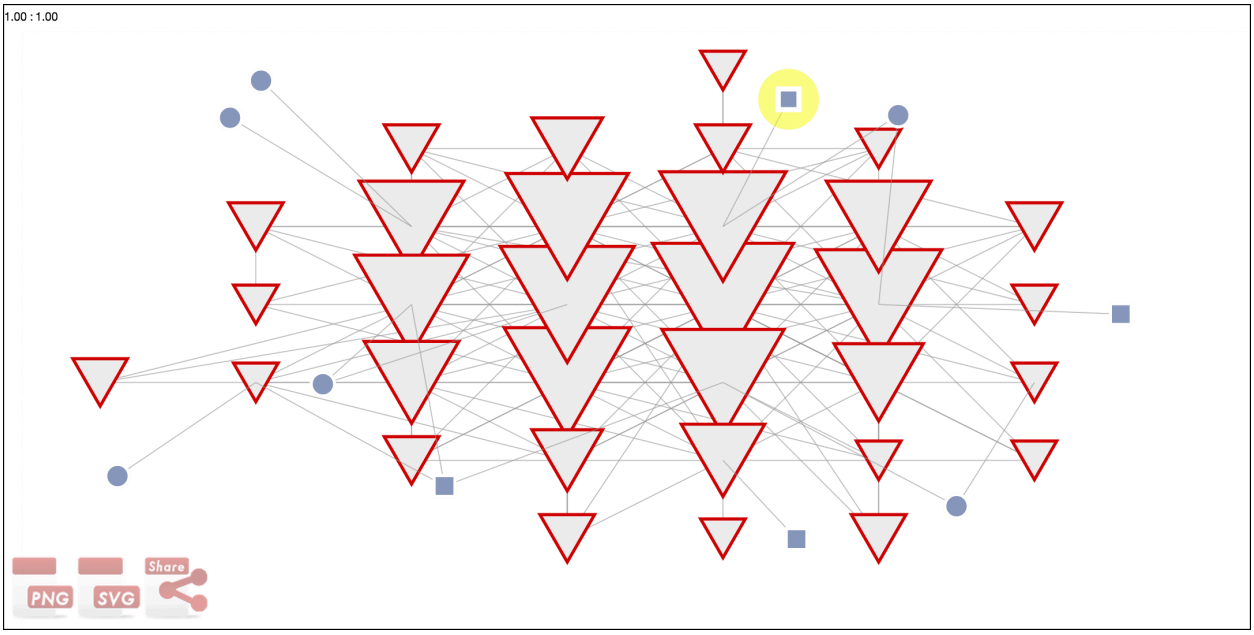


**Figure 3.2.9:** Clustering the whole dataset. Cluster representation of the dataset Platelets2014.

Figure 3.2.9 is the result of visualising the cluster representation of the Platelets2014 dataset. The user can explore the content of a cluster by selecting it, and the list of proteins is displayed in the popup widget for details. Alternatively, the user can double click on a cluster and PINV will request a deeper level of that specific cluster which in turn will be reflected by replacing that cluster by its four children in the quadtree.

## 3.3 Discussion

PINV is an open source, native web application that uses the latest generation of web technologies to offer an interactive view of protein-protein interactions which is easily accessible from any modern browser. PINV enables researchers to explore their data using different methods and the visualization can be manipulated to highlight the proteins or interactions of interest. The resulting graphic can be exported to common graphic formats, shared via URL or embedded in third-party pages, features that make it suitable for publication, sharing and collaboration activities.

We consider PINV to be a unique tool for visualization and exploration of PPI networks, not only in terms of the technology used (HTML5) but also because today's research requires collaborative efforts to process data. PINV offers an intuitive way to visualize, share and publish PPI data without the need to install any extra software (besides a web browser) and/or third party components such as Flash or Java.

The proposed method to explore whole datasets using an spatial clustering algorithm is not a complete representation of the real network, and we are aware of several implications of the implementation decisions, for example, that the partition by quadtrees can split clusters well defined that other methods can detect easily. However we consider that the use of quadtrees, gives a partial solution that is efficient and allows an easy way to explore a whole dataset form top to bottom.

*Knowing is not enough; we must apply. Willing is not enough; we must do*

Goethe

# 4

# Use Cases

*here is something*, cool to say

## 4.1 Orthologs in 3 Mycobactherium Organisms

multiorganism and filters -TODO: create links with DASty

## 4.2 Human Population Genetics

to show the clustering in action!! Maybe linking with myKaryoView

## 4.3 Shotgun Analysis of Platelets From Dengue Patients

Real research example

*Ouch*

Homero

# 5
# Conclusions

*here is something*, cool to say

# References


[1] Abigail Acland, Richa Agarwala, Tanya Barrett, Jeff Beck, Dennis A Benson, Colleen Bollin, Evan Bolton, Stephen H Bryant, Kathi Canese, Deanna M Church, et al. Database resources of the national center for biotechnology information. *Nucleic acids research*, 42(D1):D7, 2014.

[2] Giuseppe Agapito, Pietro Guzzi, and Mario Cannataro. Visualization of protein interaction networks: problems and solutions. *BMC Bioinformatics*, 14(Suppl 1):S1, 2013. ISSN 1471-2105. URL http://www.biomedcentral.com/1471-2105/14/S1/S1.

[3] Joanna S Amberger, Carol A Bocchini, François Schiettecatte, Alan F Scott, and Ada Hamosh. Omim. org: Online mendelian inheritance in man (omim®), an online catalog of human genes and genetic disorders. *Nucleic acids research*, page gku1205, 2014.

[4] Adam Ameur, Vladimir Yankovski, Stefan Enroth, Ola Spjuth, and Jan Komorowski. The lcb data warehouse. *Bioinformatics*, 22(8):1024–1026, 2006. doi: 10.1093/bioinformatics/btl036. URL http://bioinformatics.oxfordjournals.org/content/22/8/1024.abstract.

[5] Bruno Aranda, Hagen Blankenburg, Samuel Kerrien, Fiona SL Brinkman, Arnaud Ceol, Emilie Chautard, Jose M Dana, Javier De Las Rivas, Marine Dumousseau, Eugenia Galeota, et al. Psicquic and psiscore: accessing and scoring molecular interactions. *Nature methods*, 8(7): 528–529, 2011.

[6] Josh Barnes and Piet Hut. A hierarchical o(n log n) force-calculation algorithm. *Nature*, 324 (6096):446–449, 12 1986. URL http://dx.doi.org/10.1038/324446a0.

[7] C Bauer and G King. Java persistence with hibernate. *Manning Publications*, November 2006.

[8] François Belleau, Marc-Alexandre Nolin, Nicole Tourigny, Philippe Rigault, and Jean Morissette. Bio2rdf: towards a mashup to build bioinformatics knowledge systems. *Journal of biomedical informatics*, 41(5):706–716, 2008.

[9] M. Bostock, V. Ogievetsky, and J. Heer. D3; data-driven documents. *Visualization and Computer Graphics, IEEE Transactions on*, 17(12):2301–2309, 2011. ISSN 1077-2626.

[10] Christoph Brinkrolf, Sebastian Jan Janowski, Benjamin Kormeier, Martin Lewinski, Klaus Hippe, Daniela Borck, and Ralf Hofestädt. Vanesa - a software application for the visualization and analysis of networks in systems biology applications. Journal of Integrative Bioinformatics - JIB (ISSN 1613-4516), 2014.

[11] Melissa S Cline, Michael Smoot, Ethan Cerami, Allan Kuchinsky, Nerius Landys, Chris Workman, Rowan Christmas, Iliana Avila-Campilo, Michael Creech, Benjamin Gross, et al. Integration of biological networks and gene expression data using cytoscape. *Nature protocols*, 2 (10):2366–2382, 2007.

[12] Melissa S Cline, W James Kent, et al. Understanding genome browsing. *Nature biotechnology*, 27 (2):153, 2009.

[13] Niklaas Colaert, Kenny Helsens, Francis Impens, Joël Vandekerckhove, and Kris Gevaert. Rover: a tool to visualize and validate quantitative proteomics data from different sources. *Proteomics*, 10 (6):1226–1229, 2010.

[14] 1000 Genomes Project Consortium et al. An integrated map of genetic variation from 1,092 human genomes. *Nature*, 491(7422):56–65, 2012.

[15] The UniProt Consortium. Ongoing and future developments at the universal protein resource. *Nucleic Acids Research*, 39(suppl 1):D214–D219, 2011. doi: 10.1093/nar/gkq1020. URL http://nar.oxfordjournals.org/content/39/suppl_1/D214.abstract.

[16] Manuel Corpas, Rafael Jimenez, Seth J Carbon, Alex García, Leyla Garcia, Tatyana Goldberg, John Gomez, Alexis Kalderimis, Suzanna E Lewis, Ian Mulvany, Aleksandra Pawlik, Francis Rowland, Gustavo A. Salazar, Fabian Schreiber, Ian Sillitoe, William H. Spooner, Anil S. Thanki, José M. Villaveces, Guy Yachdav, and Henning Hermjakob. Biojs: an open source standard for biological visualisation–its status in 2014. *F1000Research*, 3, 2014.

[17] Fiona Cunningham, M Ridwan Amode, Daniel Barrell, Kathryn Beal, Konstantinos Billis, Simon Brent, Denise Carvalho-Silva, Peter Clapham, Guy Coates, Stephen Fitzgerald, et al. Ensembl 2015. *Nucleic acids research*, page gku1010, 2014.

[18] Noemi del Toro, Marine Dumousseau, Sandra Orchard, Rafael C Jimenez, Eugenia Galeota, Guillaume Launay, Johannes Goll, Karin Breuer, Keiichiro Ono, Lukasz Salwinski, et al. A new reference implementation of the psicquic web service. *Nucleic acids research*, 41(W1): W601–W606, 2013.

[19] Emek Demir, Michael P Cary, Suzanne Paley, Ken Fukuda, Christian Lemer, Imre Vastrik, Guanming Wu, Peter D'Eustachio, Carl Schaefer, Joanne Luciano, et al. The biopax community standard for pathway data sharing. *Nature biotechnology*, 28(9):935–942, 2010.

[20] Eric W Deutsch, Henry Lam, and Ruedi Aebersold. Data analysis and bioinformatics tools for tandem mass spectrometry in proteomics. *Physiological genomics*, 33(1):18–25, 2008.

[21] Robin Dowell, Rodney Jokerst, Allen Day, Sean Eddy, and Lincoln Stein. The distributed annotation system. *BMC Bioinformatics*, 2(1):7, 2001. ISSN 1471-2105. doi: 10.1186/1471-2105-2-7. URL http://www.biomedcentral.com/1471-2105/2/7.

[22] Thomas A Down, Matias Piipari, and Tim J. P. Hubbard. Dalliance: interactive genome viewing on the web. *Bioinformatics*, 2011. doi: 10.1093/bioinformatics/btr020. URL http://bioinformatics.oxfordjournals.org/content/early/2011/01/19/bioinformatics.btr020.abstract.

[23] Thure Etzold, Anatoly Ulyanov, and Patrick Argos. Srs: information retrieval system for molecular biology data banks. *Methods in enzymology*, 266:114, 1996.

[24] Kurt Fellenberg, Nicole C. Hauser, Benedikt Brors, Jörg D. Hoheisel, and Martin Vingron. Microarray data warehouse allowing for inclusion of experiment annotations in statistical analysis. *Bioinformatics*, 18(3):423–433, 2002. doi: 10.1093/bioinformatics/18.3.423. URL http://bioinformatics.oxfordjournals.org/content/18/3/423.abstract.

[25] Xosé M. Fernández-Suárez, Daniel J. Rigden, and Michael Y. Galperin. The 2014 nucleic acids research database issue and an updated nar online molecular biology database collection. *Nucleic Acids Research*, 42(D1):D1–D6, 2014. doi: 10.1093/nar/gkt1282. URL http://nar.oxfordjournals.org/content/42/D1/D1.abstract.

[26] Robert D. Finn, James W. Stalker, David K. Jackson, Eugene Kulesha, Jody Clements, and Roger Pettett. ProServer: a simple, extensible Perl DAS server. *Bioinformatics*, 23(12):1568–1570, 2007. doi: 10.1093/bioinformatics/btl650. URL http://bioinformatics.oxfordjournals.org/cgi/content/abstract/23/12/1568.

[27] Robert D. Finn, John Tate, Jaina Mistry, Penny C. Coggill, Stephen John Sammut, Hans-Rudolf Hotz, Goran Ceric, Kristoffer Forslund, Sean R. Eddy, Erik L. L. Sonnhammer, and Alex Bateman. The Pfam protein families database. *Nucl. Acids Res.*, 36(suppl_1):D281–288, 2008. doi: 10.1093/nar/gkm960. URL http://nar.oxfordjournals.org/cgi/content/abstract/36/suppl_1/D281.

[28] Paul Flicek, M. Ridwan Amode, Daniel Barrell, Kathryn Beal, Simon Brent, Yuan Chen, Peter Clapham, Guy Coates, Susan Fairley, Stephen Fitzgerald, Leo Gordon, Maurice Hendrix, Thibaut Hourlier, Nathan Johnson, Andreas Kahari, Damian Keefe, Stephen Keenan, Rhoda Kinsella, Felix Kokocinski, Eugene Kulesha, Pontus Larsson, Ian Longden, William McLaren, Bert Overduin, Bethan Pritchard, Harpreet Singh Riat, Daniel Rios, Graham R. S. Ritchie, Magali Ruffier, Michael Schuster, Daniel Sobral, Giulietta Spudich, Y. Amy Tang, Stephen Trevanion, Jana Vandrovcova, Albert J. Vilella, Simon White, Steven P. Wilder, Amonida Zadissa, Jorge Zamora, Bronwen L. Aken, Ewan Birney, Fiona Cunningham, Ian Dunham, Richard Durbin, Xose M. Fernandez-Suarez, Javier Herrero, Tim J. P. Hubbard, Anne Parker, Glenn Proctor, Jan Vogel, and Stephen M. J. Searle. Ensembl 2011. *Nucleic Acids Research*, 39(suppl 1):D800–D806, 2011. doi: 10.1093/nar/gkq1064. URL http://nar.oxfordjournals.org/content/39/suppl_1/D800.abstract.

[29] Andrea Franceschini, Damian Szklarczyk, Sune Frankild, Michael Kuhn, Milan Simonovic, Alexander Roth, Jianyi Lin, Pablo Minguez, Peer Bork, Christian von Mering, and Lars J. Jensen. String v9.1: protein-protein interaction networks, with increased coverage and integration. *Nucleic Acids Research*, 41(D1):D808–D815, 2013. URL http://nar.oxfordjournals.org/content/41/D1/D808.abstract.

[30] Kelly A Frazer, Lior Pachter, Alexander Poliakov, Edward M Rubin, and Inna Dubchak. Vista: computational tools for comparative genomics. *Nucleic acids research*, 32(suppl 2):W273–W279, 2004.

[31] Terrence S Furey. Comparison of human (and other) genome browsers. *Human genomics*, 2(4): 266, 2006.

[32] Nils Gehlenborg, Sean I O'Donoghue, Nitin S Baliga, Alexander Goesmann, Matthew A Hibbs, Hiroaki Kitano, Oliver Kohlbacher, Heiko Neuweger, Reinhard Schneider, Dan Tenenbaum, and Anne-Claude Gavin. Visualization of omics data for systems biology. *Nature Methods*, pages S56–68, March 2010.

[33] Bernat Gel Moreno, Andrew Jenkinson, Rafael Jimenez, Xavier Messeguer Peypoch, and Henning Hermjakob. easydas: Automatic creation of das servers. *BMC Bioinformatics*, 12(1):23, 2011. ISSN 1471-2105. doi: 10.1186/1471-2105-12-23. URL http://www.biomedcentral.com/1471-2105/12/23.

[34] Robert C Gentleman, Vincent J Carey, Douglas M Bates, Ben Bolstad, Marcel Dettling, Sandrine Dudoit, Byron Ellis, Laurent Gautier, Yongchao Ge, Jeff Gentry, et al. Bioconductor: open software development for computational biology and bioinformatics. *Genome biology*, 5(10):R80, 2004.

[35] Belinda Giardine, Cathy Riemer, Ross C Hardison, Richard Burhans, Laura Elnitski, Prachi Shah, Yi Zhang, Daniel Blankenberg, Istvan Albert, James Taylor, et al. Galaxy: a platform for interactive large-scale genome analysis. *Genome research*, 15(10):1451–1455, 2005.

[36] Carole Goble and Robert Stevens. State of the nation in data integration for bioinformatics. *Journal of Biomedical Informatics*, 41(5):687 – 693, 2008. ISSN 1532-0464. doi: DOI:10.1016/j.jbi.2008.01.008. URL http://www.sciencedirect.com/science/article/B6WHD-4RS43MK-5/2/ab521e38479b012d688a645191bfa1c6. Semantic Mashup of Biomedical Data.

[37] Jeremy Goecks, Anton Nekrutenko, James Taylor, et al. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biol*, 11(8):R86, 2010.

[38] John Gómez, Leyla J García, Gustavo A Salazar, Jose Villaveces, Swanand Gore, Alexander García, Maria J Martín, Guillaume Launay, Rafael Alcántara, Noemi Del Toro Ayllón, Marine Dumousseau, Sandra Orchard, Sameer Velankar, Henning Hermjakob, Chenggong Zong, Peipei Ping, Manuel Corpas, and Rafael C Jiménez. Biojs: An open source javascript framework for biological data visualization. *Bioinformatics*, 2013. URL http://bioinformatics.oxfordjournals.org/content/early/2013/02/23/bioinformatics.btt100.abstract.

[39] Asia Grzibovska and Andreas Prlic. DAS2 writeback server implementation. Master's thesis, Chalmers University of Technology, 2008.

[40] Henning Hermjakob. The hupo proteomics standards initiative–overcoming the fragmentation of proteomics data. *Proteomics*, 6(S2):34–38, 2006.

[41] Klaus Hippe, Benjamin Kormeier, Thoralf Töpel, Sebastian Janowski, and Ralf Hofestädt. Dawis-m.d. - a data warehouse system for metabolic data. In *Conference: Informatik 2010: Service Science - Neue Perspektiven für die Informatik*. Jahrestagung der Gesellschaft für Informatik, 2010.

[42] Robert Hoffmann. A wiki for the life sciences where authorship matters. *Nature genetics*, 40(9): 1047–1051, 2008.

[43] Danny Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *Visualization and Computer Graphics, IEEE Transactions on*, 12(5):741–748, 2006.

[44] Zhenjun Hu, Yi-Chien Chang, Yan Wang, Chia-Ling Huang, Yang Liu, Feng Tian, Brian Granger, and Charles DeLisi. Visant 4.0: Integrative network platform to connect genes, drugs, diseases and therapies. *Nucleic Acids Research*, 41(W1):W225–W231, 2013. URL http://nar.oxfordjournals.org/content/41/W1/W225.abstract.

[45] Duncan Hull, Katy Wolstencroft, Robert Stevens, Carole Goble, Mathew R Pocock, Peter Li, and Tom Oinn. Taverna: a tool for building and running workflows of services. *Nucleic acids research*, 34(suppl 2):W729–W732, 2006.

[46] Sarah Hunter, Rolf Apweiler, Teresa K. Attwood, Amos Bairoch, Alex Bateman, David Binns, Peer Bork, Ujjwal Das, Louise Daugherty, Lauranne Duquenne, Robert D. Finn, Julian Gough, Daniel Haft, Nicolas Hulo, Daniel Kahn, Elizabeth Kelly, Aur√©lie Laugraud, Ivica Letunic, David Lonsdale, Rodrigo Lopez, Martin Madera, John Maslen, Craig McAnulla, Jennifer McDowall, Jaina Mistry, Alex Mitchell, Nicola Mulder, Darren Natale, Christine Orengo, Antony F. Quinn, Jeremy D. Selengut, Christian J. A. Sigrist, Manjula Thimma, Paul D. Thomas, Franck Valentin, Derek Wilson, Cathy H. Wu, and Corin Yeats. Interpro: the integrative protein signature database. *Nucleic Acids Research*, 37(suppl 1):D211–D215, 2009. doi: 10.1093/nar/gkn785. URL http://nar.oxfordjournals.org/content/37/suppl_1/D211.abstract.

[47] Curtis Huttenhower, Sajid Mehmood, and Olga Troyanskaya. Graphle: Interactive exploration of large, dense graphs. *BMC Bioinformatics*, 10(1):417, 2009.

[48] Richard J Jacob. Bioinformatics for lc-ms/ms-based proteomics. In *LC-MS/MS in Proteomics*, pages 61–91. Springer, 2010.

[49] Andrew Jenkinson, Mario Albrecht, Ewan Birney, Hagen Blankenburg, Thomas Down, Robert Finn, Henning Hermjakob, Tim Hubbard, Rafael Jimenez, Philip Jones, Andreas Kahari, Eugene Kulesha, Jose Macias, Gabrielle Reeves, and Andreas Prlic. Integrating biological data - the distributed annotation system. *BMC Bioinformatics*, 9(Suppl 8):S3, 2008. ISSN 1471-2105. doi: 10.1186/1471-2105-9-S8-S3. URL http://www.biomedcentral.com/1471-2105/9/S8/S3.

[50] Rafael C. Jimenez, Antony F. Quinn, Alexander Garcia, Alberto Labarga, Kieran O'Neill, Fernando Martinez, Gustavo A. Salazar, and Henning Hermjakob. Dasty2, an Ajax protein DAS

client. *Bioinformatics*, 24(18):2119–2121, 2008. doi: 10.1093/bioinformatics/btn387. URL http://bioinformatics.oxfordjournals.org/cgi/content/abstract/24/18/2119.

[51] Rafael C Jimenez, Gustavo A Salazar, Bernat Gel, Joaquin Dopazo, Nicola Mulder, and Manuel Corpas. mykaryoview: a light-weight client for visualization of genomic data. *PloS one*, 6(10): e26345, 2011.

[52] Philip Jones, Nisha Vinod, Thomas Down, Andre Hackmann, Andreas Kahari, Ernst Kretschmann, Antony Quinn, Daniela Wieser, Henning Hermjakob, and Rolf Apweiler. Dasty and uniprot das: a perfect pair for protein feature visualization. *Bioinformatics*, 21(14): 3198–3199, 2005.

[53] M Aleksi Kallio, Jarno T Tuimala, Taavi Hupponen, Petri Klemelä, Massimiliano Gentile, Ilari Scheinin, Mikko Koski, Janne Käki, and Eija I Korpelainen. Chipster: user-friendly analysis software for microarray and other high-throughput data. *BMC genomics*, 12(1):507, 2011.

[54] Arek Kasprzyk. Biomart: driving a paradigm change in biological data management. *Database*, 2011, 2011. doi: 10.1093/database/bar049. URL http://database.oxfordjournals.org/content/2011/bar049.short.

[55] Thomas Kelder, Martijn P van Iersel, Kristina Hanspers, Martina Kutmon, Bruce R Conklin, Chris T Evelo, and Alexander R Pico. Wikipathways: building research communities on biological pathways. *Nucleic acids research*, 40(D1):D1301–D1307, 2012.

[56] W. J. Kent, A. S. Zweig, G. Barber, A. S. Hinrichs, and D. Karolchik. Bigwig and bigbed: enabling browsing of large distributed datasets. *Bioinformatics*, 26(17):2204–2207, 2010. doi: 10.1093/bioinformatics/btq351. URL http://bioinformatics.oxfordjournals.org/content/26/17/2204.abstract.

[57] Samuel Kerrien, Bruno Aranda, Lionel Breuza, Alan Bridge, Fiona Broackes-Carter, Carol Chen, Margaret Duesbury, Marine Dumousseau, Marc Feuermann, Ursula Hinz, Christine Jandrasits, Rafael C. Jimenez, Jyoti Khadake, Usha Mahadevan, Patrick Masson, Ivo Pedruzzi, Eric Pfeiffenberger, Pablo Porras, Arathi Raghunath, Bernd Roechert, Sandra Orchard, and Henning Hermjakob. The intact molecular interaction database in 2012. *Nucleic Acids Research*, 40(D1): D841–D846, 2012.

[58] Haim Kilov. From semantic to object-oriented data modeling. In *ISCI '90: Proceedings of the first international conference on systems integration on Systems integration '90*, pages 385–393, Piscataway, NJ, USA, 1990. IEEE Press. ISBN 0-8186-9027-5.

[59] Rhoda J. Kinsella, Andreas Kahari, Syed Haider, Jorge Zamora, Glenn Proctor, Giulietta Spudich, Jeff Almeida-King, Daniel Staines, Paul Derwent, Arnaud Kerhornou, Paul Kersey, and Paul Flicek. Ensembl biomarts: a hub for data retrieval across taxonomic space. *Database*, 2011, 2011. doi: 10.1093/database/bar030. URL http://database.oxfordjournals.org/content/2011/bar030.abstract.

[60] Rafał Kuć. *Apache Solr 4 Cookbook*. PACKT Publishing, 2013.

[61] Michael Kuhn, Christian von Mering, Monica Campillos, Lars Juhl Jensen, and Peer Bork. Stitch: interaction networks of chemicals and proteins. *Nucleic Acids Research*, 36(suppl 1):D684–D688, 2008. URL http://nar.oxfordjournals.org/content/36/suppl_1/D684.abstract.

[62] Eduardo Lee, Gregg A Helt, Justin T Reese, Monica C Munoz-Torres, Chris P Childers, Robert M Buels, Lincoln Stein, Ian H Holmes, Christine G Elsik, and Suzanna E Lewis. Web apollo: a web-based genomic annotation editing platform. *Genome Biol*, 14:R93, 2013.

[63] Thomas Lee, Yannick Pouliot, Valerie Wagner, Priyanka Gupta, David Stringer-Calvert, Jessica Tenenbaum, and Peter Karp. Biowarehouse: a bioinformatics database warehouse toolkit. *BMC Bioinformatics*, 7(1):170, 2006. ISSN 1471-2105. doi: 10.1186/1471-2105-7-170. URL http://www.biomedcentral.com/1471-2105/7/170.

[64] Heng Li, Bob Handsaker, Alec Wysoker, Tim Fennell, Jue Ruan, Nils Homer, Gabor Marth, Goncalo Abecasis, Richard Durbin, and 1000 Genome Project Data Processing Subgroup. The sequence alignment/map format and samtools. *Bioinformatics*, 25(16):2078–2079, 2009. doi: 10.1093/bioinformatics/btp352. URL http://bioinformatics.oxfordjournals.org/content/25/16/2078.abstract.

[65] Wenlin Li, Lisa N. Kinch, and Nick V. Grishin. Pclust: protein network visualization highlighting experimental data. *Bioinformatics*, 29(20):2647–2648, 2013. doi: 10.1093/bioinformatics/btt451. URL http://bioinformatics.oxfordjournals.org/content/29/20/2647.abstract.

[66] Luana Licata, Leonardo Briganti, Daniele Peluso, Livia Perfetto, Marta Iannuccelli, Eugenia Galeota, Francesca Sacco, Anita Palma, Aurelio Pio Nardozza, Elena Santonico, Luisa Castagnoli, and Gianni Cesareni. Mint, the molecular interaction database: 2012 update. *Nucleic Acids Research*, 40(D1):D857–D861, 2012. doi: 10.1093/nar/gkr930. URL http://nar.oxfordjournals.org/content/40/D1/D857.abstract.

[67] Christian T. Lopes, Max Franz, Farzana Kazi, Sylva L. Donaldson, Quaid Morris, and Gary D. Bader. Cytoscape web: an interactive web-based network browser. *Bioinformatics*, 26(18): 2347–2348, 2010. URL http://bioinformatics.oxfordjournals.org/content/26/18/2347.abstract.

[68] Samad Lotia, Jason Montojo, Yue Dong, Gary D Bader, and Alexander R Pico. Cytoscape app store. *Bioinformatics*, page btt138, 2013.

[69] Qibin Luo, Philipp Pagel, Baiba Vilne, and Dmitrij Frishman. Dima 3.0: domain interaction map. *Nucleic acids research*, 39(suppl 1):D724–D729, 2011.

[70] Mark D McDowall, Michelle S Scott, and Geoffrey J Barton. Pips: human protein–protein interaction prediction database. *Nucleic acids research*, 37(suppl 1):D651–D656, 2009.

[71] Sheldon J McKay, Ismael A Vergara, and Jason E Stajich. Using the generic synteny browser (gbrowse_syn). *Current protocols in bioinformatics*, pages 9–12, 2010.

[72] David N. Messina and Erik L. L. Sonnhammer. DASher: a stand-alone protein sequence client for DAS, the Distributed Annotation System. *Bioinformatics*, 25(10):1333–1334, 2009. doi: 10.1093/bioinformatics/btp153. URL http://bioinformatics.oxfordjournals.org/cgi/content/abstract/25/10/1333.

[73] Harald Mischak, Rolf Apweiler, Rosamonde E Banks, Mark Conaway, Joshua Coon, Anna Dominiczak, Jochen HH Ehrich, Danilo Fliser, Mark Girolami, Henning Hermjakob, et al. Clinical proteomics: a need to define the field and to begin to set adequate standards. *PROTEOMICS-Clinical Applications*, 1(2):148–156, 2007.

[74] Alex Mitchell, Hsin-Yu Chang, Louise Daugherty, Matthew Fraser, Sarah Hunter, Rodrigo Lopez, Craig McAnulla, Conor McMenamin, Gift Nuka, Sebastien Pesseat, et al. The interpro protein families database: the classification resource after 15 years. *Nucleic acids research*, page gku1243, 2014.

[75] Martin Morgan and Fred Hutchinson. Bioconductor annual report. Annual report, Bioconductor, July 2014.

[76] Cydney B Nielsen, Michael Cantor, Inna Dubchak, David Gordon, and Ting Wang. Visualizing genomes: techniques and challenges. *Nature methods*, 7:S5–S15, 2010.

[77] Yun Niu, David Otasek, and Igor Jurisica. Evaluation of linguistic features useful in extraction of interactions from pubmed; application to annotating known, high-throughput and predicted interactions in i2d. *Bioinformatics*, 26(1):111–119, 2010.

[78] National Library of Medicine (US). *Genetics Home Reference - How do geneticists indicate the location of a gene?* The Library; National Library of Medicine, 2013. URL http://ghr.nlm.nih.gov/handbook/howgeneswork/genelocation.

[79] Utkan Ogmen, Ozlem Keskin, A Selim Aytuna, Ruth Nussinov, and Attila Gursoy. Prism: protein interactions by structural matching. *Nucleic acids research*, 33(suppl 2):W331–W336, 2005.

[80] Yoshiyuki Ohtsubo, Wakako Ikeda-Ohtsubo, Yuji Nagata, and Masataka Tsuda. Genomematcher: a graphical user interface for dna sequence comparison. *BMC bioinformatics*, 9 (1):376, 2008.

[81] Alexandre Panchaud, Michael Affolter, Philippe Moreillon, and Martin Kussmann. Experimental and computational approaches to quantitative proteomics:< i> status quo</i> and outlook. *Journal of proteomics*, 71(1):19–33, 2008.

[82] Cesare Pautasso, Olaf Zimmermann, and Frank Leymann. Restful web services vs. "big"' web services: making the right architectural decision. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, pages 805–814, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-085-2. doi: http://doi.acm.org/10.1145/1367497.1367606.

[83] Georgios Pavlopoulos, Anna-Lynn Wegener, and Reinhard Schneider. A survey of visualization tools for biological network analysis. *BioData Mining*, 1(1):12, 2008. ISSN 1756-0381. URL http://www.biodatamining.org/content/1/1/12.

[84] Yasset Perez-Riverol, Rui Wang, Henning Hermjakob, Markus Müller, Vladimir Vesada, and Juan Antonio Vizcaíno. Open source libraries and frameworks for mass spectrometry based proteomics: A developer's perspective. *Biochimica et Biophysica Acta (BBA)-Proteins and Proteomics*, 1844(1):63–76, 2014.

[85] Alexander R Pico, Thomas Kelder, Martijn P Van Iersel, Kristina Hanspers, Bruce R Conklin, and Chris Evelo. Wikipathways: pathway editing for the people. *PLoS biology*, 6(7):e184, 2008.

[86] Katja Pratsch, Robert Wellhausen, and Harald Seitz. Advances in the quantification of protein microarrays. *Current opinion in chemical biology*, 18:16–20, 2014.

[87] Andreas Prlic, Thomas A. Down, and Tim J. P. Hubbard. Adding Some SPICE to DAS. *Bioinformatics*, 21(suppl_2):ii40–41, 2005. doi: 10.1093/bioinformatics/bti1106. URL http://bioinformatics.oxfordjournals.org/cgi/content/abstract/21/suppl_2/ii40.

[88] Andreas Prlic, Thomas Down, Eugene Kulesha, Robert Finn, Andreas Kahari, and Tim Hubbard. Integrating sequence and structural biology with DAS. *BMC Bioinformatics*, 8(1):333, 2007. ISSN 1471-2105. doi: 10.1186/1471-2105-8-333. URL http://www.biomedcentral.com/1471-2105/8/333.

[89] James T Robinson, Helga Thorvaldsdóttir, Wendy Winckler, Mitchell Guttman, Eric S Lander, Gad Getz, and Jill P Mesirov. Integrative genomics viewer. *Nature biotechnology*, 29(1):24–26, 2011.

[90] Kate R Rosenbloom, Joel Armstrong, Galt P Barber, Jonathan Casper, Hiram Clawson, Mark Diekhans, Timothy R Dreszer, Pauline A Fujita, Luvina Guruvadoo, Maximilian Haeussler, et al. The ucsc genome browser database: 2015 update. *Nucleic acids research*, page gku1177, 2014.

[91] Tome S Silva, Nadege Richard, Jorge P Dias, and Pedro M Rodrigues. Data visualization and feature selection methods in gel-based proteomics. *Current Protein and Peptide Science*, 15(1): 4–22, 2014.

[92] Rintaro Saito, Michael E Smoot, Keiichiro Ono, Johannes Ruscheinski, Peng-Liang Wang, Samad Lotia, Alexander R Pico, Gary D Bader, and Trey Ideker. A travel guide to cytoscape plugins. *Nature methods*, 9(11):1069–1076, 2012.

[93] Gustavo Salazar, Nicola Mulder, and Edwin Blake. DAS Writeback: A Collaborative Annotation System. Master's thesis, University of Cape Town, 2010. URL http://pubs.cs.uct.ac.za/archive/00000609/.

[94] Gustavo Salazar, Rafael Jimenez, Alexander Garcia, Henning Hermjakob, Nicola Mulder, and Edwin Blake. Das writeback: A collaborative annotation system. *BMC Bioinformatics*, 12(1):143, 2011. ISSN 1471-2105. doi: 10.1186/1471-2105-12-143. URL http://www.biomedcentral.com/1471-2105/12/143.

[95] Gustavo A. Salazar, Leyla J. García, Philip Jones, Rafael C. Jimenez, Antony F. Quinn, Andrew M. Jenkinson, Nicola Mulder, Maria Martin, Sarah Hunter, and Henning Hermjakob. Mydas, an extensible java das server. *PLoS ONE*, 2012.

[96] Gustavo A Salazar, Ayton Meintjes, Gaston Mazandu, Holifidy A Rapanoël, Richard O Akinola, and Nicola J Mulder. A web-based protein interaction network visualizer. *BMC bioinformatics*, 15 (1):129, 2014.

[97] Gustavo A. Salazar, Ayton Meintjes, and Nicola Mulder. Ppi layouts: Biojs components for the display of protein-protein interactions [v1; ref status: awaiting peer review, http://f1000r.es/2u5]. *F1000Research*, 3-50, 2014.

[98] Lukasz Salwinski, Christopher S Miller, Adam J Smith, Frank K Pettit, James U Bowie, and David Eisenberg. The database of interacting proteins: 2004 update. *Nucleic acids research*, 32(suppl 1): D449–D451, 2004.

[99] Martin H Schaefer, Jean-Fred Fontaine, Arunachalam Vinayagam, Pablo Porras, Erich E Wanker, and Miguel A Andrade-Navarro. Hippie: Integrating protein interaction networks with experiment based quality scores. *PloS one*, 7(2):e31826, 2012.

[100] Michael C Schatz, Adam M Phillippy, Ben Shneiderman, and Steven L Salzberg. Hawkeye: an interactive visual analytics tool for genome assemblies. *Genome biology*, 8(3):R34, 2007.

[101] Thomas Schmitt, Christoph Ogris, and Erik LL Sonnhammer. Funcoup 3.0: database of genome-wide functional coupling networks. *Nucleic acids research*, 42(D1):D380–D388, 2014.

[102] Sohrab Shah, Yong Huang, Tao Xu, Macaire Yuen, John Ling, and BF Francis Ouellette. Atlas - a data warehouse for integrative bioinformatics. *BMC Bioinformatics*, 6(1):34, 2005. ISSN 1471-2105. doi: 10.1186/1471-2105-6-34. URL http://www.biomedcentral.com/1471-2105/6/34.

[103] Paul Shannon, Andrew Markiel, Owen Ozier, Nitin S. Baliga, Jonathan T. Wang, Daniel Ramage, Nada Amin, Benno Schwikowski, and Trey Ideker. Cytoscape: A software environment for integrated models of biomolecular interaction networks. *Genome Research*, 13(11):2498–2504, 2003.

[104] Ben Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Visual Languages, 1996. Proceedings., IEEE Symposium on*, pages 336–343. IEEE, 1996.

[105] Mitchell E Skinner, Andrew V Uzilov, Lincoln D Stein, Christopher J Mungall, and Ian H Holmes. Jbrowse: a next-generation genome browser. *Genome research*, 19(9):1630–1638, 2009.

[106] Michael E Smoot, Keiichiro Ono, Johannes Ruscheinski, Peng-Liang Wang, and Trey Ideker. Cytoscape 2.8: new features for data integration and network visualization. *Bioinformatics*, 27(3): 431–432, 2011.

[107] Giulietta M Spudich and Xosé M Fernández-Suárez. Touring ensembl: a practical guide to genome browsing. *BMC genomics*, 11(1):295, 2010.

[108] James Stalker, Brian Gibbins, Patrick Meidl, James Smith, William Spooner, Hans-Rudolf Hotz, and Antony V Cox. The ensembl web site: mechanics of a genome browser. *Genome research*, 14 (5):951–955, 2004.

[109] Chris Stark, Bobby-Joe Breitkreutz, Teresa Reguly, Lorrie Boucher, Ashton Breitkreutz, and Mike Tyers. Biogrid: a general repository for interaction datasets. *Nucleic acids research*, 34(suppl 1): D535–D539, 2006.

[110] Matthew Suderman and Michael Hallett. Tools for visually exploring biological networks. *Bioinformatics*, 23(20):2651–2659, 2007.

[111] Damian Szklarczyk, Andrea Franceschini, Stefan Wyder, Kristoffer Forslund, Davide Heller, Jaime Huerta-Cepas, Milan Simonovic, Alexander Roth, Alberto Santos, Kalliopi P Tsafou, et al. String v10: protein–protein interaction networks, integrated over the tree of life. *Nucleic acids research*, page gku1003, 2014.

[112] Thoralf Topel, Benjamin Kormeier, Andreas Klassen, and Ralf Hofestadt. Biodwh: a data warehouse kit for life science data integration. *J Integr Bioinform*, 5(2), 2008. ISSN 1613-4516 (Electronic); 1613-4516 (Linking). doi: 10.2390/biecoll-jib-2008-93.

[113] Ben P Vandervalk, E Luke McCarthy, and Mark D Wilkinson. Moby and moby 2: creatures of the deep (web). *Briefings in bioinformatics*, 10(2):114–128, 2009.

[114] Jose M Villaveces, Rafael C Jimenez, Leyla J Garcia, Gustavo A Salazar, Bernat Gel, Nicola Mulder, Maria Martin, Alexander Garcia, and Henning Hermjakob. Dasty3, a web framework for das. *Bioinformatics*, 27(18):2616–2617, 2011.

[115] Steve Vinoski. Serendipitous reuse. *IEEE Internet Computing*, 12(1):84–87, 2008. ISSN 1089-7801. doi: http://dx.doi.org/10.1109/MIC.2008.20.

[116] Juan Antonio Vizcaino, Richard Cote, Florian Reisinger, Joseph M. Foster, Michael Mueller, Jonathan Rameseder, Henning Hermjakob, and Lennart Martens. A guide to the proteomics identifications database proteomics data repository. *PROTEOMICS*, 9(18):4276–4283, 2009. ISSN 1615-9861. doi: 10.1002/pmic.200900402. URL http://dx.doi.org/10.1002/pmic.200900402.

[117] Rui Wang, Antonio Fabregat, Daniel Ríos, David Ovelleiro, Joseph M Foster, Richard G Côté, Johannes Griss, Attila Csordas, Yasset Perez-Riverol, Florian Reisinger, et al. Pride inspector: a tool to visualize and validate ms proteomics data. *Nature biotechnology*, 30(2):135–137, 2012.

[118] Rui Wang, Yasset Perez-Riverol, Henning Hermjakob, and Juan Antonio Vizcaíno. Open source libraries and frameworks for biological data visualisation: A guide for developers. *Proteomics*, 2014.

[119] David Warde-Farley, Sylva L Donaldson, Ovi Comes, Khalid Zuberi, Rashad Badrawi, Pauline Chao, Max Franz, Chris Grouios, Farzana Kazi, Christian Tannus Lopes, et al. The genemania

prediction server: biological network integration for gene prioritization and predicting gene function. *Nucleic acids research*, 38(suppl 2):W214–W220, 2010.

[120] Mark Wilkinson, Benjamin Vandervalk, and Luke McCarthy. The semantic automated discovery and integration (sadi) web service design-pattern, api and reference implementation. *Journal of Biomedical Semantics*, 2(1):8, 2011. ISSN 2041-1480. doi: 10.1186/2041-1480-2-8. URL http://www.jbiomedsem.com/content/2/1/8.

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

[122] Katherine Wolstencroft, Robert Haines, Donal Fellows, Alan Williams, David Withers, Stuart Owen, Stian Soiland-Reyes, Ian Dunlop, Aleksandra Nenadic, Paul Fisher, et al. The taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud. *Nucleic acids research*, page gkt328, 2013.

[123] Evgeni M. Zdobnov, Rodrigo Lopez, Rolf Apweiler, and Thure Etzold. The ebi srs server—recent developments. *Bioinformatics*, 18(2):368–373, 2002. doi: 10.1093/bioinformatics/18.2.368. URL http://bioinformatics.oxfordjournals.org/content/18/2/368.abstract.

[124] Qiangfeng Cliff Zhang, Donald Petrey, José Ignacio Garzón, Lei Deng, and Barry Honig. Preppi: a structure-informed database of protein–protein interactions. *Nucleic acids research*, page gks1231, 2012.

[125] Zhang Zhang, Vladimir B. Bajic, Jun Yu, Kei-Hoi Cheung, and Jeffrey P. Townsend. *Trends and Methodologies*, chapter 2. Data Integration in Bioinformatics: Current Efforts and Challenges, Bioinformatics. Intech, November 2011. URL http://www.intechopen.com/books/bioinformatics-trends-and-methodologies/data-integration-in-bioinformatics-current-efforts-and-challenges.

# A

# Results of a stress test between DAS servers.

# Results of a stress test between DAS servers.

We prepared a stress test using the Apache HTTP Server Benchmarking Tool (http://httpd.apache.org/docs/2.0/programs/ab.html) to compare the loading performance of DAS servers.
Three servers were installed on the same machine: MyDas, ProServer and Dazzle.
The requests were triggered on the same machine, with the purpose of avoiding any bias due to network conditions and computer specs.

The test was designed to repeat the same query 1000 times with 10 concurrent connections. Three queries were used in the test, one to return a document of approximately 1500 bytes(small), a second one with 200000 bytes(medium), and one returning the whole file, which in DAS format is approximately 7200000 bytes(large).

**Specifications of the machine used for the test**

- Brand: Apple
- Model: Macbook pro a1211
- Processor: 2.16 GHz intel Core 2 Duo
- Memory: 3 GB 667MHz DDR2 SDRAM
- Operative System: Mac OS X 10.6.8 Snow Leopard

**Data**

The data set used for the tests corresponds with the annotations for the mitochondrial DNA of the organism *C. Elegans* freely available at the URL:
ftp://ftp.sanger.ac.uk/pub/wormbase/releases/WS220/genomes/c_elegans/genome_feature_tables/GFF2/CHROMOSOME_MtDNA.gff
The file contains 18510 annotations and is 3.4 MB in size. It is a text file whose format follows the GFF2 standard.

**Software Installation**

MyDas and Dazzle are Java servlet applications. The servlet container used for both was Apache Tomcat 7.0.11 and the Java Runtime Environment running on the machine is version 1.6.0_33.
ProServer requires perl and the version installed on the test machine is version v5.12.3.
The installation instructions of the software and its corresponding data sources can be found on the relevant server web sites:

- MyDas:
  http://code.google.com/p/mydas/wiki/MyDASTutorial
- ProServer:
  http://proserver.svn.sourceforge.net/viewvc/proserver/trunk/doc/proserver_guide.html#install
- Dazzle:
  http://biojava.org/wiki/Dazzle#Getting_Dazzle

**Test execution and result files:**

The results returned by the apache benchmarking tool during testing are shown below

- **Small test** (~1500bytes):
  - MyDas

```
Server Software:        Apache-Coyote/1.1
Server Hostname:        localhost
```

```
Server Port:            8080

Document Path:          /MyDasTemplate/das/examplegff/features?segment=CHROMOSOME_MtDNA:1,50
Document Length:        1554 bytes

Concurrency Level:      10
Time taken for tests:   1.352 seconds
Complete requests:      1000
Failed requests:        0
Write errors:           0
Total transferred:      2124000 bytes
HTML transferred:       1554000 bytes
Requests per second:    739.88 [#/sec] (mean)
Time per request:       13.516 [ms] (mean)
Time per request:       1.352 [ms] (mean, across all concurrent requests)
Transfer rate:          1534.68 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    1   1.1      0      13
Processing:     2   13  11.2     10     112
Waiting:        0   12  11.1      9     112
Total:          2   13  11.3     10     112

Percentage of the requests served within a certain time (ms)
  50%     10
  66%     14
  75%     18
  80%     20
  90%     27
  95%     36
  98%     47
  99%     53
 100%    112 (longest request)
```

- ProServer

```
Server Software:        ProServer/756
Server Hostname:        localhost
Server Port:            9000

Document Path:          /das/mygff/features?segment=CHROMOSOME_MtDNA:1,50
Document Length:        1421 bytes

Concurrency Level:      10
Time taken for tests:   649.298 seconds
Complete requests:      1000
Failed requests:        0
Write errors:           0
Total transferred:      1865000 bytes
HTML transferred:       1421000 bytes
Requests per second:    1.54 [#/sec] (mean)
Time per request:       6492.978 [ms] (mean)
Time per request:       649.298 [ms] (mean, across all concurrent requests)
Transfer rate:          2.81 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   5.9      0     186
Processing:  4256 6481 403.0   6453   11716
Waiting:     4255 6478 403.1   6447   11715
Total:       4256 6481 403.1   6453   11716

Percentage of the requests served within a certain time (ms)
  50%   6453
  66%   6621
  75%   6726
  80%   6782
  90%   6947
  95%   7064
  98%   7244
```

```
  99%   7353
 100%  11716 (longest request)
```

- Dazzle

```
Server Software:        Apache-Coyote/1.1
Server Hostname:        localhost
Server Port:            8080

Document Path:          /dazzle-webapp/tss/features?segment=CHROMOSOME_MtDNA:1,50
Document Length:        1605 bytes

Concurrency Level:      10
Time taken for tests:   2.355 seconds
Complete requests:      1000
Failed requests:        0
Write errors:           0
Total transferred:      2074000 bytes
HTML transferred:       1605000 bytes
Requests per second:    424.56 [#/sec] (mean)
Time per request:       23.554 [ms] (mean)
Time per request:       2.355 [ms] (mean, across all concurrent requests)
Transfer rate:          859.91 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    1   1.7      0      22
Processing:     3   23  26.4     14     196
Waiting:        0   21  25.4     13     196
Total:          3   23  26.4     14     197

Percentage of the requests served within a certain time (ms)
  50%     14
  66%     21
  75%     27
  80%     32
  90%     52
  95%     80
  98%    115
  99%    135
 100%    197 (longest request)
```

- **Medium test** (~20000bytes):
  - MyDas

```
Server Software:        Apache-Coyote/1.1
Server Hostname:        localhost
Server Port:            8080

Document Path:          /MyDasTemplate/das/examplegff/features?segment=CHROMOSOME_MtDNA:1,200
Document Length:        216986 bytes

Concurrency Level:      10
Time taken for tests:   19.411 seconds
Complete requests:      1000
Failed requests:        0
Write errors:           0
Total transferred:      217556000 bytes
HTML transferred:       216986000 bytes
Requests per second:    51.52 [#/sec] (mean)
Time per request:       194.114 [ms] (mean)
Time per request:       19.411 [ms] (mean, across all concurrent requests)
Transfer rate:          10944.96 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   1.1      0      19
Processing:    19  193 159.6    155    1305
Waiting:       11  180 161.5    138    1265
Total:         19  194 159.6    156    1305
```

```
Percentage of the requests served within a certain time (ms)
  50%    156
  66%    203
  75%    242
  80%    268
  90%    350
  95%    482
  98%    795
  99%    876
 100%   1305 (longest request)
```

- ProServer

```
Server Software:        ProServer/756
Server Hostname:        localhost
Server Port:            9000

Document Path:          /das/mygff/features?segment=CHROMOSOME_MtDNA:1,200
Document Length:        209768 bytes

Concurrency Level:      10
Time taken for tests:   712.340 seconds
Complete requests:      1000
Failed requests:        0
Write errors:           0
Total transferred:      210214000 bytes
HTML transferred:       209768000 bytes
Requests per second:    1.40 [#/sec] (mean)
Time per request:       7123.396 [ms] (mean)
Time per request:       712.340 [ms] (mean, across all concurrent requests)
Transfer rate:          288.19 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.4      0      10
Processing:  5646 7116 220.2   7129    7836
Waiting:     5645 7112 220.5   7125    7835
Total:       5646 7116 220.2   7129    7836

Percentage of the requests served within a certain time (ms)
  50%   7129
  66%   7191
  75%   7228
  80%   7254
  90%   7336
  95%   7423
  98%   7612
  99%   7689
 100%   7836 (longest request)
```

- Dazzle

```
Server Software:        Apache-Coyote/1.1
Server Hostname:        localhost
Server Port:            8080

Document Path:          /dazzle-webapp/tss/features?segment=CHROMOSOME_MtDNA:1,200
Document Length:        209656 bytes

Concurrency Level:      10
Time taken for tests:   29.322 seconds
Complete requests:      1000
Failed requests:        0
Write errors:           0
Total transferred:      210103000 bytes
HTML transferred:       209656000 bytes
Requests per second:    34.10 [#/sec] (mean)
Time per request:       293.216 [ms] (mean)
Time per request:       29.322 [ms] (mean, across all concurrent requests)
Transfer rate:          6997.52 [Kbytes/sec] received
```