DAS Writeback: A Collaborative Annotation System for Proteins

Gustavo A. Salazar O.

· May 2010 ·

Department of Computer Science
University of Cape Town

Submitted in partial fulfilment of the requirements for the degree of Master of Science

Supervised by
Prof. Edwin Blake

Co-supervised by
Dr. Nicola Mulder

This work was supported by the National Bioinformatics Network of South Africa

"A hundred times every day I remind myself that my inner and outer life depend on the labors of other men, living and dead, and that I must exert myself in order to give in the same measure as I have received and am still receiving." – *Albert Einstein*

Abstract

We designed and developed a Collaborative Annotation System for Proteins called DAS Writeback, which extends the Distributed Annotation System (DAS) to provide the functionalities of adding, editing and deleting annotations.
A great deal of effort has gone into gathering information about proteins over the last few years. By June 2009, UniProtKB/Swiss-Prot, a curated database, contained over four hundred thousand sequence entries and UniProtKB/TrEMBL, a database with automated annotation, contained over eight million sequence entries. Every protein is annotated with relevant information, which needs to be efficiently captured and made available to other research groups. These include annotations about the structure, the function or the biochemical residues.
Several research groups have taken on the task of making this information accessible to the community, however, information flow in the opposite direction has not been extensively explored. Users are currently passive actors that behave as consumers of one or several sources of protein annotations and they have no immediate way to provide feedback to the source if, for example, a mistake is detected or they want to add information. Any change has to be done by the owner of the database. The current lack of being able to feed information back to a database is tackled in this project.
The solution consists of an extension of the DAS protocol that defines the communication rules between the client and the writeback server following the Uniform Interface of the RESTful architecture. A protocol extension was proposed to the DAS community and implementations of both server and client were created in order to have a fully functional system. For the development of the server, writing functionalities were added to MyDAS, which is a widely used DAS server. The writeback client is an extended version of the web-based protein client Dasty2.
The involvement of the DAS community and other potential users was a fundamental component of this project. The architecture was designed with the insight of the DAS specialized forum, a prototype was then created and subsequently presented in the DAS workshop 2009. The feedback from the forum and workshop was used to redefine the architecture and implement the system. A usability experiment was performed using potential users of the system emulating a real annotation task. It demonstrated that DAS writeback is effective, usable and will provide the appropriate environment for the creation and evolution of a protein annotation community.
Although the scope of this research is limited to protein annotations, the specification was defined in a general way. It can, therefore, be used for other types of information supported by DAS, implying that the server is versatile enough to be used in other scenarios without major modifications.

Contents

List of Figures

List of Tables

Acknowledgements

I want to use this space to recognize all the people who have contributed in so many different ways to this project. Let me start with my supervisors; Edwin and Nicky have shared their knowledge during the last two years. I really appreciate your guidance and the valuable insight you put in this project.

Rafael was the Spanish guy married to a Chinese girl living in South Africa who helped this Colombian guy to survive his first days in a new country. His advice and ideas were a very important part of this project. Another big contribution of ideas and recommendations were from Alex who, although I found his character undecipherable, seemed almost always willing to help me. Elizabeth was the reason why this document was written in pretty decent English. Without her, the spanglish on it would it have made it unreadable.

To all my friends, both the new and old ones, to those who motived me from Colombia, to those that adopted me as the new friend in South Africa, to those who played Capoeira with me, to those with whom I worked with in the same Lab, to those with whom I shared so many good times, to all you guys, Many Many Thanks!

Last but not least, to my family; their unconditional support can be felt even with the Atlantic between us. Mamá, Papá y Juancho you are the most important people to me and it's your love that gives me energy to work harder every time.

Muchas Gracias!!!

Cape Town, January 2010

List of Abbreviations and Acronyms

BO Biosapiens Ontology

CSHL Cold Spring Harbor Laboratory

CRUD Create, Read, Update and Delete

DAS Distributed Annotation System

DiLAS Digital Library Annotation Service

DL Digital Libraries

DNA Deoxyribonucleic Acid

DOI Document Object Identifier

DOM Document Object Model

ECO Evidence Code Ontology

EMBL European Molecular Biology Laboratory

FAST Flexible Annotation Service Tool

GFF General Feature Format

HTTP HyperText Transfer Protocol

JSP Java Server Pages

JWS Java Web Start

LDAS Lightweight Distributed Annotation Server

MADCOW Multimedia Annotation of Digital Content Over the Web

MIME Multipurpose Internet Mail Extensions

MVC Model View Control pattern

NCBI National Center for Biotechnology Information

REST REpresentational State Transfer

RNA Ribonucleic Acid

SW Semantic Web

PDB Protein Data Bank

PFAM Protein Family Database

RSS Really Simple Syndication

SOAP Simple Object Access Protocol

SQL Standard Query Language

SVN Subversion

UCD User Cetered Design

UniProtKB UniProt Knowledgebase

URI Universal Resource Identifier

URL Universal Resource Locator

XML eXtended Markup Language

WWW World Wide Web

Chapter 1

Introduction

The annotation of biological data is a common task in different fields of the life sciences. For example, a taxonomist can annotate that one of the differences between two species of insects is how far evolved their eyes and antennae are, and an ecologist may be interested in annotations about the population of a certain ecosystem. For bioinformaticians, the raw materials are the digital representations of different genes and their products (DNA, RNA, proteins, etc.) and, therefore, any information about the sequence, experiment, genetic material, etc., becomes an annotation.

In the central dogma of biology, DNA (deoxyribonucleic acid) encodes genes, which get translated in 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).

The annotation methods for genetic material can be classified into manual and automatic [11].

Manual annotation refers to the actions of an individual, usually an expert in the field, annotating the evidence extracted during a review of published scientific literature. It is a valuable effort that produces important outcomes like UniProtKB/Swiss-Prot, a manually annotated database of high quality protein information.

Automatic annotation works under a similarity hypothesis, for instance when two very similar sequences (homologues) have a common ancestor, then their functions and features

should be the same. Therefore, any annotation in one of the sequences can be extrapolated to the other. Automatic annotation is required because of the flood of data; genome projects, among others, are able to generate terabytes of information on a daily basis and it is therefore impossible to have enough experts to process this amount of data. However, automatic annotation is dangerous [7], because it can infer erroneous features and these can be propagated.

A balance between the two types of annotation is required in order to deal simultaneously with the massive sets of biological data and with the details where the similarity approach can generate errors. Manual annotation then becomes a way to polish the information obtained by automatic methods.

Stein identified four organizational models to describe the way that genomic annotation is done [46]. The *Factory* model is highly automated; it is applied in the first stages of the annotation looking for the location of genes and/or protein domains. In contrast, the *Museum* model is widely used in the latter stages and is mainly manual. It is done by experts and is focused on the function of regions that have already been detected. An alternative model is the *Cottage*, where experts dedicate time out of their regular activities to a specific project. And finally, the *Party* model, as an extension of the *Cottage* model, puts all that extra time of the experts together in an intensive period where all of them are in the same place with a specific annotation task.

None of these models consider the option of obtaining information from the final users of the system. The Web, as an example, has grown exponentially during the last years, in part thanks to this principle. Authors and readers have started to mix their roles in what is now known as Web 2.0. Following the analogy of organizational models, we propose the *Tourism* model; here the tourist can enjoy the features of a site, but sporadically can also add new features to the site or modify the existing ones.

Tourists can be experts or not, and their contribution to the annotations is completely voluntary. It is then possible to find tourists that are purely consumers, which request information but do not add or modify anything in the knowledge base. In contrast, other kinds of tourist are also expected, including those that are more proactive and willing to share their knowledge on a particular protein. They give new annotations or correct existing ones, enriching the information for themselves but also for the purely consumer tourists.

A collaborative model such as the one described here can bring together the critical mass of experts needed to improve the quality of the automatic annotations. It will, however, also generate an issue of confidence in the annotators on the part of the users.

Having annotations captured using different models, with different techniques and by different laboratories, creates the problem of how to integrate all the information. The Distributed Annotation System, DAS [12], proposed a standard to publish and query annotations of several types of genetic material/product, like DNA and proteins. It works under the idea of federated databases where the information is distributed in several places and each of those is specialized in a particular subtopic. In that way, for example, a laboratory specialized in the 3D configuration of a protein can provide annotation about protein structures, and other institutions focused on the function of proteins can provide functional information. A DAS client can then read the features from both places for the same protein and put them in the same context.

DAS offers a practical solution for compiling information from different sources in a single client; however the current state of the protocol and therefore the implementations of the servers, do not provide a method to send the information in the opposite direction– i.e. the user generating information about current or new annotations for a particular protein.

We believe that this bidirectional communication will enrich the knowledge database that DAS has conglomerated with its federated approach. On one hand, there is a clear advantage in providing tools to final users in order to get feedback and to enrich the source with the knowledge of the user contributions. On the other hand, the owners of the information invest a great deal of energy in creating and consolidating the information and therefore are not willing to allow careless users or users with bad intentions to jeopardize their efforts.

1.1 Research Question

To be concise, this project aims to answer the following question; *"How can the current DAS infrastructure be extended in order to capture and use information provided by the users?"*. This problem is represented graphically in the figure 1.1 where the green arrow represents the current direction of the communications in DAS, and the red arrow represents the one proposed by this project.

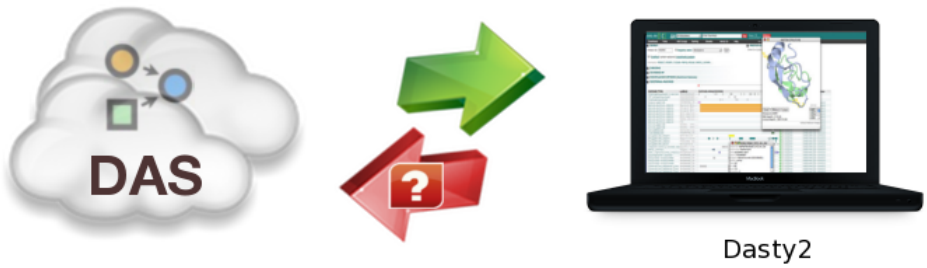

Figure 1.1: *Research Question*: How can the current DAS infrastructure be extended in order to capture and use information provided by the users?

1.2 Approach

The solution to this problem involves an extension of the architectural model of DAS to support the necessary writing features for a Collaborative Annotation System, with the definition of the specifications for the writeback capabilities following the same logic and style of the DAS protocol. Implementations of client and server were also created as proof of concept of the proposed architecture. The main method of experimental Computer Science is to build artefacts and then evaluate them experimentally [47]. This document describes all the details and required steps for the creation of a complete solution to the given research problem. An experiment to test the usability of the client, and indirectly the effectiveness of the whole system, was also executed.

We proposed the addition of a new server into the DAS system, called writeback. This server should be responsible for the management of the information that users want to introduce into the system. The writeback server should be independent of any other DAS server and it should reference the sources without modifying the original data.

The communication methods of the new server with other components of DAS can be defined using several strategies– in this project we used two of these. The first one was based on the writeback document included in the DAS2.0 specification. DAS2.0 was not widely adopted and the idea of replacing DAS with it was abandoned (Section 2.3.2). It pushed us to create our own strategy and a protocol extension was proposed to be included in the future official version of DAS.

The proposed protocol consists of incorporating the Uniform Interface feature of RESTful

services into DAS, using the HTTP[1] methods to indicate the desired operation, and the same XML[2] format defined in the DAS specification to envelope the information to send from the client to the server.

Implementations of both strategies were developed and a discussion about the pros and cons of each is included in Chapter 4.

We consider that a collaborative system requires easy-to-use tools that users can intuitively manipulate, and for this reason, we looked for a widely adopted DAS client, conserving its user interface methods and extending it to support the communication with the writeback server. Between several clients we chose to extend Dasty2, a web based protein DAS client, which, given its features of extensibility and usability, was adequate for the requirement of a writeback client (Details in Chapter 5)

To ensure the compatibility with all the components of DAS, getting feedback about the advances, and ideas to solve arising issues during the process, a continuous communication with the DAS community was held through email lists and also by presenting the progress of the project at the DAS workshop that is held once a year.

When we reached the point of having a functional prototype, we proceeded to execute a formative evaluation, in order to detect usability issues, but more importantly, to verify that the whole system was doing what users are expecting.

The involvement of experts during the definition of the protocol and architecture, plus their feedback to overcome issues arising during the project, and the execution of evaluations with untrained users, are consequence of our user-centred design approach.

The organization of the content of this document can be found in the following road map.

1.3 Road map

Chapter 2 covers the general background of the project, highlighting the topics of special relevance for the development of the project.

[1] *HTTP*: The Hypertext Transfer Protocol is the set of rules for exchanging files (text, graphic images, sound, video, and other multimedia files) on the World Wide Web.

[2] *XML* Extensible Markup Language; a flexible text format for creating structured computer documents

Chapter 3 includes the definition of the architectural extension proposed for DAS and the discussion about two alternatives for the writeback communication protocol.

Chapter 4 covers the implementation of two DAS writeback servers, dscussing which of those should be included in the Collaborative Annotation System.

Chapter 5 covers the design and implementation of the DAS writeback client, including snapshots of the final implementation.

Chapter 6 covers the design, execution and analysis of the usability experiment, indicating the errors and suggestions captured by the experiment and the changes made to the software.

In Chapter 7 we draw conclusions from the whole project.

Chapter 2

Background

2.1 Introduction

In order to clearly understand the details of this project, it is necessary to have background knowledge on some general topics: Annotations, the Distributed Annotation System and User Centred Design.

Annotation is a tool that has been used for a long time in very different fields, and in recent years its use has exploded due to the increase in web sites where most of the content is created by users.

A particular field that has used annotations is Digital Libraries; the more relevant considerations in the field for this project are included here.

Bioinformatics should not be excluded from the use of annotations, projects such as wikiproteins and gene-wiki are also described here.

The Distributed Annotation System provides an environment for taking advantage of the distributed nature of biological information. The details of this protocol, including its architectural behavior, are described in the second part of this chapter.

The importance of clients and servers in the DAS architecture is then described. It lays out the main implementation of both server and client, highlighting the features that can

have an impact on the Collaborative Protein Annotation System.

Afterwards, a brief introduction on User Centred Design and some of the usability evaluation techniques is presented in order to define some important concepts of our approach.

2.2 Annotation

The task of annotating is not new at all; terms like *gloss*, *scholium* and *postil* have been used for centuries to refer to different types of annotation. As reported in [2] these words have their origin in ancient Greek or Latin, indicating that the process of annotating documents or artifacts is as old as the antique cultures themselves.

In current times annotations are linked to advances in the field of information technology, and we are now increasingly more likely to find spaces to comment on a particular resource on the Web. That is the case with two of the most successful web sites, www.youtube.com and www.flickr.com, where you can rate, comment or answer a video or image respectively. In other words, you have many different methods to annotate a web resource.

In an effort to standardize the annotation of web resources, the Annotea project [27] creates some metaphors to improve collaborative environments in the Web using Semantic Web[1] (SW) technologies as its basis. As more users utilize these tools, more metadata will be created for the SW and vice versa. There are two client implementations of this idea: one is a Firefox plug-in called Ubimarks and the other is called Amaya for Mozilla.

It is not surprising that the web is becoming a platform for annotation, especially since earlier documents about the web include the annotation of resources as one of the goals. For example [6] describes the now historical browser Mosaic. An extract from this document said about Mosaic: *"Asynchronous collaboration capabilities, including text and voice annotations for documents located anywhere on the Internet. These annotations can exist at either the private or workgroup level."* The big difference is that the scale, amount of data, number of people and computational power involved are just so much larger now.

Annotations have also been a focus of research in the field of Digital Libraries (DL). Here,

[1] *Semantic Web*: Is the evolving project of the WWW where the meaning of the information is defined, making it possible for machines to process it.

the annotations have become a very important way to share knowledge between the users of the DL. A current project called DiLAS [4] is looking for a standardized way to share annotations between different Digital Library Management Systems.

A key feature of all these briefly described annotation systems is their collaborative character. An individual annotation is important for just one person, but when the annotation is shared in a public environment, such as a digital library or the web, this annotation can be complemented, corrected or discussed by other users and, therefore, over time the annotations become a source of information as important as the annotated resource itself.

Probably the most important collaborative environment developed during the last few years is www.wikipedia.org. Wikipedia is a multilingual, web-based, encyclopedia project; it has an open philosophy, allowing anyone to edit content. Currently, it has more than 2,000,000 articles entirely added by its users, demonstrating how powerful a collaborative approach can become. This open strategy raises the issue of how trustworthy the source of the information is, and several studies have been done on this topic, for instance, in [1] a system is proposed where the authors of the content have a reputation, and this reputation depends on the number of consistent edits by that author.

Collaborative trustworthy systems are very important in scientific environments because scientists require the most recent and trustworthy information in order to get good results in their research. For the purposes of this project we will focus on annotations in bioinformatics, and more specifically on proteins.

2.2.1 Annotations in Digital Libraries

Digital Libraries DL have been using annotations over the last few years. As is reported in [4], different systems have been developed to organize this information in a way that allows users to create new information in the context of a resource of the DL. Therefore, the concept of annotation has been widely studied in this context. For example a formal model for annotation is defined in [3]. The model was created using set theory and defines an annotation in terms of the annotated document, the user or group that is doing the annotating, the type, part and the meaning of the annotation. It also involves the concepts of time, permissions and scope.

A study examining the habit of annotating textbooks in a public university library and its implications for the adoption of annotation systems on DL is included in [32]. From there, the next group of suggestions about annotations in DL was selected given its relevance for this project: *In situ annotation, distinguishable from the source*, *Smooth transitions between public and private annotations* and the *Integration with reading as an activity.* Other suggestions included in the study were not relevant to this project because they were more book-oriented.

A Web 2.0 oriented set of decision points for annotations in DL can be found in [16]. Besides the items that are similar to those already selected from [32], the following are the most relevant decision points for our project: *Ease of annotation*, *Control of content*, *Ease of retrieval* and *Notification and sharing*

A clear example of the use of annotations in DL is the *Digital Library Annotation Service DiLAS* [4]; a user interface to create and visualize annotations in DL. The idea was to re-use two technologies: The *Flexible Annotation Service Tool FAST* and the *Multimedia Annotation of Digital Content Over the Web MADCOW.*

The classic three-layer architecture is the base of FAST and it is reused in DiLAS. The novel idea of FAST is to be independent of any DLMS, each resource has a universal identifier as a URI or a DOI. In this way the annotation could be shared, even between different *Digital Library Management System DLMS.*

MADCOW is a client-server system, not related to DL, but that allows creating annotations over web documents. DiLAS allows for annotations coming from a MADCOW system, related to the resources of a FAST system.

2.2.2 Annotations in Bioinformatics

Stein defines genome annotation as *"The process of taking the raw DNA sequence produced by the genome-sequencing projects and adding the layers of analysis and interpretation necessary to extract its biological significance and place into the context of our understanding of biological processes"*. He classifies the annotations into three levels: Nucleotide, Protein and Process. In the nucleotide level, the focus is *where* the information is, for example where a gene is located in the genome, or where a non-coding region is. Annotating pro-

teins is about the *what*, as in, what is the result of the transcription of a gene, or what kind of protein is the one to annotate. Annotation of the process is also called Functional annotation. It had its breakthrough point with the release of Gene Ontology, because it became the standard between projects for integrating this kind of information. [46].

In general, the search for integration platforms has been a big challenge in the different areas of bioinformatics. Consortiums such as the International Nucleotide Sequence Database Collaboration INSDC (DDBJ, EMBL, and GenBank)[9] are a perfect example of the big effort that has been made to integrate and share biological information. In the case of biological annotations, the Distributed Annotation System has established a set of standards to make the annotations from different servers available in the same context. However additional data that adds value to these resources needs a simple and rapid route to public access. A collaborative approach similar to www.wikipedia.org can be an alternative to reach this goal.

The idea of using wiki-based techniques in scientific fields is not new, for instance wikiproteins [36] is a project that enables community annotation of biological concepts. The goal of the wikiproteins project is to collect facts from the literature in order to enable collaborative knowledge discovery.

Gene Wiki is a similar project to wikiproteins was developed for genes in San Diego, California [49]. They built a system that loads data from Entrez Gene database to build wiki pages with the most relevant information in this database, and creating hyperlinks to other sources of information such as the Protein Data Bank or Ensembl. As with any other wiki-page, any user of the system can update these, and in that way the creators of Gene Wiki expect that its users would add more information about the genes.

2.3 The Distributed Annotation System (DAS)

The Distributed Annotation System (DAS) [12] 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 provides 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. The

next section provides an overview of a regular DAS transaction, from when a user sends a query to when a result is returned to the user.

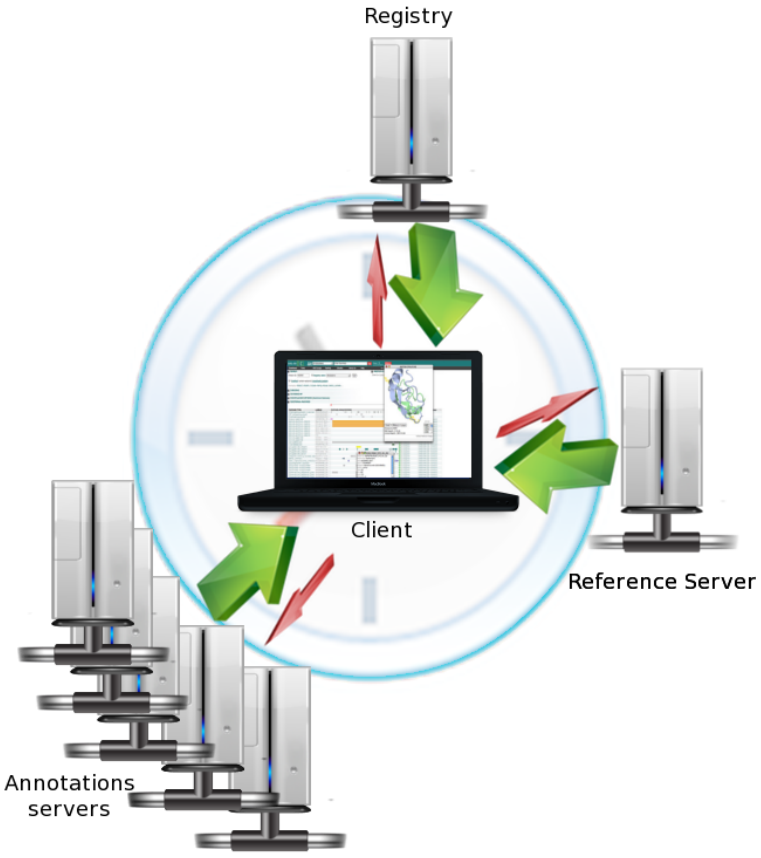

Figure 2.1: *DAS architecture:* The graphic represents the communication and its order in time of the most important entities that participate in a regular DAS query. There is a clock in the background implying the chronological order of the calls. The client is the responsible for the orchestration, calling and processing the information of the Registry, Reference and Annotation servers.

2.3.1 Behavioral description of the architecture

Figure 2.1 represents a high level view of the architecture of DAS. The red arrows represent the requests and the green arrows the responses. The width of the arrows indicates how much information is being transferred. The transaction starts when a user makes a query

using an accession number[2]. The client queries the DAS registry to find out which servers are available for proteins. With this information, it is possible to get the reference sequence[3] and basic information about the protein target from the reference server. Finally, a request is sent to all the annotation servers for features of the reference. The big challenge for the client is to merge all this information in a comprehendible and meaningful way.

2.3.2 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 [12], 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 in 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 [23]. The E in the version number is for Extended, which essentially describes the purpose of this version, because it keeps most of the features presented in 1.53 but extends these to 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[4] and it contained a redefinition of the protocol for the capabilities that DAS had in its previous versions

[2] *Unique identifier to access its information in a biological Database. For example a UniProt ID for proteins*

[3] *Reference Sequence*: The concensus sequence to refer for all the annotation sources in DAS

[4] http://biodas.org/documents/das2/das2_protocol.html 2006

(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 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.

As explained in [41], DAS follows the paradigm of the REpresentational State Transfer (REST). However, DAS has not adopted all the RESTful features. In version 1.53 and even in the draft of the 1.6 version, DAS only made use of the GET method in order to recover the information from the different servers; the other 3 methods are simply ignored in those specifications. The explanation lies in the fact that DAS sources are the owners of the information and it is not usually convenient that external users are able to modify or delete anything in its databases. As explained before, one of the strategies to solve this issue is to have the writeback server as an independent server that manages the changes, additions and deletions as meta-annotations. It is, therefore, useful if the interface for a server with these features keeps the same principles of DAS (or REST to a bigger extent).

A Protein Annotation in DAS

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 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 way, the types of annotations that can be used– as 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 into 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.

2.3.3 DAS Servers

There are several kinds of server in DAS; some serving the reference sequence, others providing information about the style of a feature, others mapping information between different coordinate systems[5]. The most common kind of DAS server, however, is the one that provides the features for a given reference, which is called an *Annotation Server*.

[5]A *Coordinate System*. Now it is a unique 4-tuple *(Authority, Version, Type, Organism)*, where *Authority* refers to the name of the institution that defines the identifiers of the system, *Version* is an optional

An annotation server in DAS has the responsibility of providing the information from a data source following the DAS specification. This implies that the original data source can be in any format, from plain files to elaborated databases. To do this, the owners of the information can develop a script from scratch that takes the information from the source to put it into the DASGFF format and publish it in the Web.

This approach has the result that any new data source has to replicate development efforts, such as the parsing or the http interface that DAS specifies. For that reason, several projects have proposed alternatives to DAS servers that incorporate the common tasks, and for the implementation of the specific details of a particular source.

Next are descriptions of the most representative DAS server implementations.

LDAS

The Lightweight Distributed Annotation Server (LDAS) is a minimalist DAS server developed at the Cold Spring Harbor Laboratory. LDAS provides the basic framework to serve the annotations following the DAS 1.53 specification. It is Perl software designed to run in Apache as a web server using a predefined MySQL database that can be loaded from tab-delimited files [29].

Having a predesigned database has the advantage of ensuring that all the annotations are in the same format; however it restricts the sources to the ones that follow the format of the files or requires extra development to convert the information to such a format.

LDAS can serve annotations, reference sequences and style-sheets, and allows the use of the DAS commands for entry points, dsn and types.

---

field to identify different assemblies of the same coordinate system, *Type* identifies the different kinds of data, like chromosome, protein sequence, etc. and finally the *Organism* field allows for the association of a coordinate system with a specific organism. As there are types that are shared in different organisms, this last field is also optional. [44]

Pro-server

Pro-server is a simple, lightweight, Perl-based DAS server that does not depend on a separate HTTP server. It is a project held by the Welcome Trust Sanger Institute since 2003 and it has been updated with the different versions of DAS. The current version of Pro-server is the only server that implements the new features of the DAS specification 1.60. [13].

Pro-server has been used to serve the genomic annotations of ENSEMBL, Gene3D and CBS, among others, which is a practical proof of its good performance.

The way that this server deals with the different data sources is through the definition of transport adaptors. There are several SourceAdaptor implementations provided with Pro-server.

Dazzle

Under the umbrella of the BioJava project the DAS server called Dazzle [20] was created. This Open-source project was developed at the Welcome Trust Sanger Institute.

Dazzle implements the version 1.53E of the DAS protocol in an architecture based on Java Servlets. In order to deal with the heterogeneity of the potential sources, Dazzle follows the plugins paradigm: a specific source should use the appropriate plugin depending on its characteristics or, in the case that there is not a plugin that suits the particular data source, this can be developed and added to Dazzle.

The available plugins for Dazzle include reading annotations from files in EMBL, UniProt and GFF formats plus a connection with databases that follow the format defined by LDAS.

In order to implement a new plugin in Dazzle, it is necessary to create a class that implements a number of interfaces depending on which DAS commands needs to be available.

MyDas

MyDas [26] is a project created and developed for EBI researchers. A stable version of MyDas is used to provide annotations in the DAS format to the UniProt database, which is one of the most important protein databases. This demonstrates the robustness of MyDas.

Its architecture is inspired by the Model-View-Control (MVC) pattern [25], MyDas has implemented a set of classes that provide the data model to build the necessary information to serve the DAS1.53 commands.

The view component of the MVC pattern is as simple as providing the results in the DAS XML format and answering the requests using the HTTP codes defined in DAS 1.5 (200:OK,400: Bad command, etc.).

The control part is more interesting; a Java servlet receives the HTTP request, identifies the command and the data source, and it is the data source which should process the command. This is because data sources in MyDas are classes that implement one of the provided interfaces and are registered in the MyDas configuration file.

With this strategy, MyDas achieves the goal of providing a standard way to process requests and responses in a DAS server, but leaves the task of dealing with the actual database to whoever implements the interface. The final storage system is, therefore, not the concern of the MyDas server.

2.3.4 DAS Clients

Dowels [12] said that a DAS client (or Annotation viewer) *"is a lightweight application whose behavior is analogous to a web browser. The viewer communicates with the genome and annotation servers using a well defined language specification"*. This definition has been evolving since 2001; at its creation, DAS was conceived only for genome annotation but now several Coordinate Systems have been created for other kinds of data including proteins, 3D structures, microarrays[6], etc. Moreover, a client can now interact with the registry to discover the different sources for a coordinate system. It also gets information

[6] *Microarray*: Chip that contains thousands of microscopic spots of DNA, used for different experiments, for example to measure changes in expression levels

on the look and feel of the annotations using the stylesheet command or makes use of the DAS ontologies for filtering purposes– just to name some of the features that the current version(1.53E) of the protocol provides [23].

Although there are a variety of functions that a DAS client could execute, the following is a list of the core tasks:

1. To query the DAS registry in order to find available sources for the specific kind of data (*Coordinate System*) that the client is interested in (proteins, genes, structure, etc.).
2. To query the reference server to get the consensus entity (sequence, structure, array, etc.) that has been requested by the user, plus some meta-information of that reference (length, version, etc.).
3. To query all the annotation sources that provide annotation for the particular coordinate system, extracting information such as the position, type, category, etc.
4. To render all this information in a single meaningful view.

There are currently implementations of DAS clients that execute those tasks in a successful way, with different flavours for different tastes. There are some Web-based clients like ENSEMBL, Dasty2 or PFAM; there are also other stand-alone applications such as Spice, Strap or DASher. The first group has the advantage of being available wherever an Internet connection is available and, given the potential familiarity that users have with the web, the learning process for those applications can be quick and almost intuitive. On the other hand, the stand-alone applications are known for having better tools for visualization purposes, making it possible to use local sources in the same context as the DAS sources. However, these implicit benefits of stand-alone applications are now less perceptible thanks to the recent development of Ajax frameworks and modern web browsers with better processing capabilities [15].

A more biologically oriented classification of the clients can be achieved using their Coordinate Systems. A group of clients that were developed to query annotations of the Coordinate Systems with *Protein Sequence* as a *Type* include: Spice, Dasty2, PFAM and DASher. ENSEMBL allows users to visualize annotations of coordinate systems with *Type*

equal to *Chromosome*, *Gene_ID* and *Contigs*. There are other DAS clients with specific purposes, such as Protein Integration (DASmi) and protein alignment (STRAP), amongst others.

The scope of this project is limited to the annotation of proteins, therefore the extension of the DAS client to include writeback capabilities should be part the first group of clients. Next is a brief description of the clients in this group.

Spice

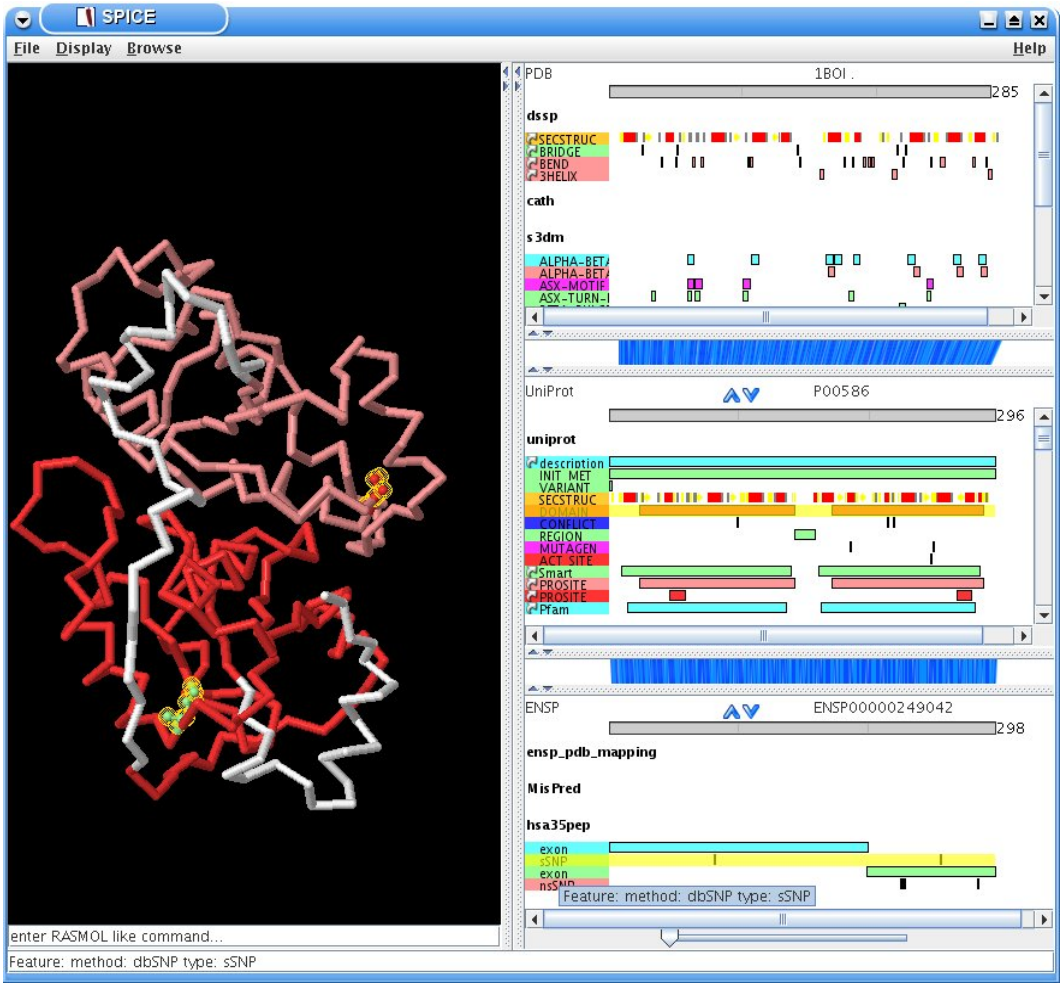

Figure 2.2: *Spice*: Java based DAS client, the interface allows for the visualization of both protein and structure annotations.

Spice is a DAS client for the visualization of protein features and structure annotations.

It is a Java stand-alone application that can run from the Internet using Java Web Start. A search in Spice can either start from the structure with a Protein Data Bank (*PDB*) accession number to find its annotations and the related protein sequences or the other way around, requesting protein information with a UniProt id to get its annotations and the related protein structures. In order to get this dual functionality, Spice makes use of a capability included in the 1.53E version of the protocol called *alignment*. An alignment server can be queried using one coordinate system and it returns a mapping between the query and its corresponding part (if it exists) in a different coordinate system [42].

Figure 2.2 is a screenshot of Spice. This interface has a set of panels for different purposes. On the left side it is possible to visualize the selected 3D structure, and the panels on the right side are, from top to bottom: the structure sequence from the PDB and its DAS annotations, the alignment with the UniProt protein sequence and its DAS annotations and finally the alignment with chromosome information from ENSEMBL, and its annotations. Although it is only possible to visualize one entity per coordinate system at a time, the user has the control to choose between the alternative alignments.

PFAM DAS client

The Protein Family Database (PFAM) project has as a main goal 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.

PFAM is not exclusively based on DAS technology, however, their aim to provide information in an easy and accessible way made this group interested in the use of the DAS protocol. For this reason, they created three data sources using Pro-server; one for domain annotations, another for sequence features as active sites and transmembrane regions, and finally one for seed and full alignments [14].

The PFAM group have also worked on a DAS visualizer especially for their annotations that allows users to get information from other sources and to put it in the same context. This client is web based and is available in the PFAM web page [21]. Figure 2.3 shows the result of a query in the PFAM DAS 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; and finally all the features of external

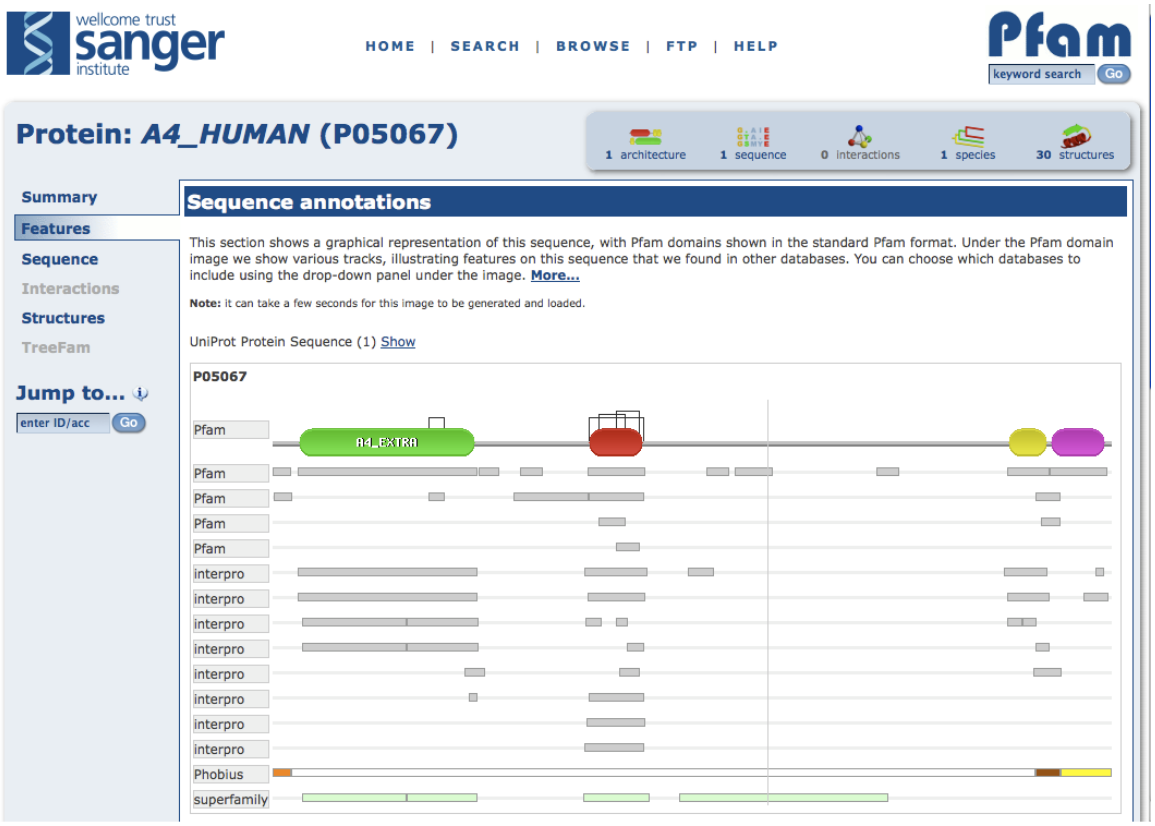

Figure 2.3: *PFAM*: Web based DAS client, which places special emphasis on the visualization of the annotations of protein domains

data sources that the user has selected for that query.

DASher

DASher is an open source protein DAS client developed in Java. It is a stand-alone application that can be run through a Java Web Start link. This project has been run by the Stockholm Bioinformatics Centre of the University of Stockholm, Sweden [33].

Figure 2.4 is a snapshot of DASher. As with other clients, detailed information of an annotation can be obtained by putting the mouse cursor over the target feature. One of the features that makes DASher different to other clients is the level of zoom, which allows the user to visualize an amino-acid detail or see the whole protein. Another advantage of DASher is that it recognizes when a source is providing continuous data and then visualizes

Figure 2.4: *DASher*: Java based DAS client, allows to visualize continuos annotation as a line plot

this information as a line plot– as can be seen at the bottom of the figure 2.4. The colour of the features can also be changed by the user. Finally, given the stand-alone nature of this client, it allows the user to save preferences locally as the order of the tracks. It is also possible to export graphics to the clipboard.

Dasty2

Dasty2 is a web-based protein DAS client, which makes extensive use of Ajax in order to make the user's experience as close as possible to using a stand-alone client.

Technology that allows for the use of Ajax has been on the main browsers (Internet explorer, Firefox, Safari, etc.) since 2002; however it was only when Google started using it in 2005 in its applications– initially it was Google Suggest[7] and then Gmail[8] and Google Maps[9]– that it became popular and started changing the look. Now almost all of the Google applications are Ajax based. Dasty2 is the DAS client that uses Ajax the most, to the point that it only requires a server proxy to walk around the Ajax constraint of just querying the same

[7] http://www.google.com/webhp?complete=1&hl=en 2005
[8] http://gmail.com 2009
[9] http://maps.google.com 2009

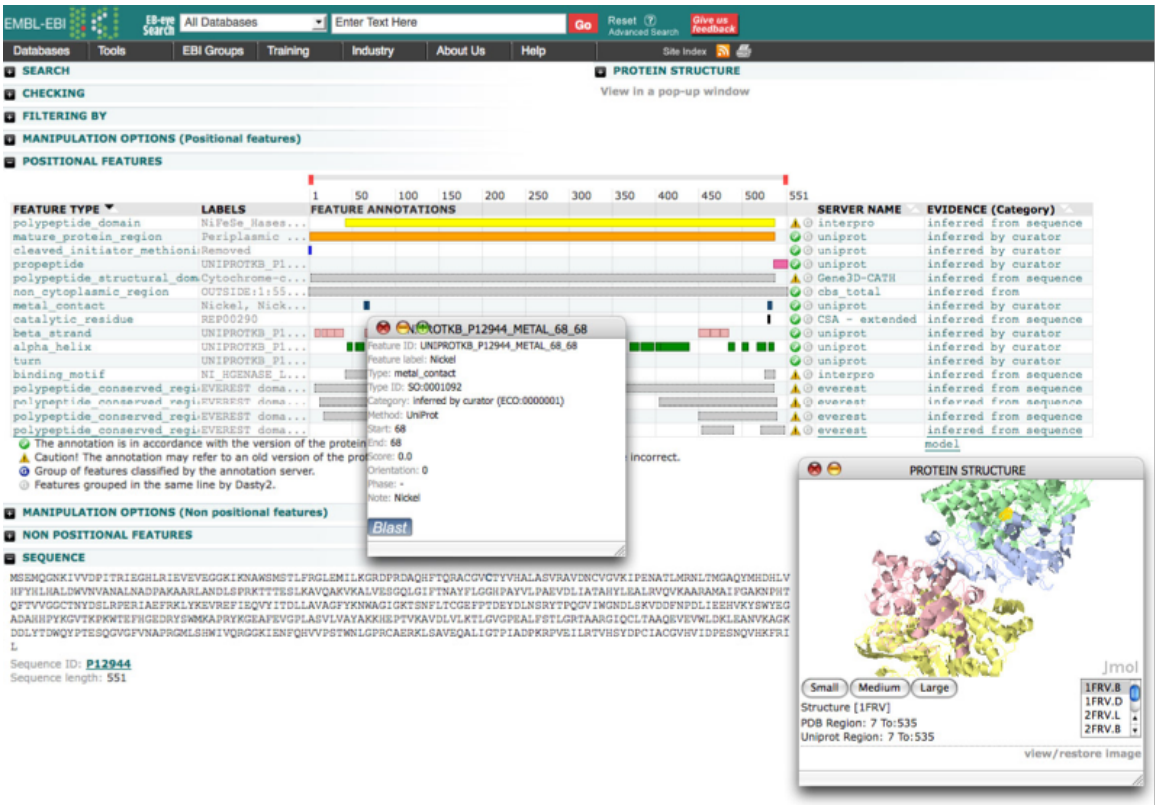

Figure 2.5: *Dasty2*: Web based DAS client, its modular interface allows the user to select between different manipulation options.

server where the Javascript is located. Obviously a DAS client requires that other servers are queried, therefore a server component is required to behave as a proxy that calls all DAS servers on request of the client. Besides that, all the functionalities run on the client, which creates the potential to have a more interactive relationship with the user, because every change that the user makes in the visualization of the graphic happens in the browser and doesn't require that the whole page is refreshed.

Figure 2.5 is a snapshot of Dasty2, in which it is possible to see some of the panels that are part of the Dasty2 interface. The division of panels is the strategy this client uses to organize all the information that is accessible from Dasty. Each panel is a module with a different function, making it easier to extend Dasty2.

The main panels of Dasty2 are as follows: *Search* is where the user introduces the protein to query, here is also possible to choose the Registry Label[10] associated with the search

[10] *Registry Label*: The DAS Registry groups annotation sources that are somehow related, could be from

in order to filter the servers to query; the *Status* panel contains the status of current processes and a list of the server requested an its response; the *Sequence* panel is where the amino acid sequence is displayed; the *Positional features* panel is the graphic where the annotations are drawn in context with the protein length; in the *Configuration* panel the user has control of the graphic characteristics such as the zoom or which columns are displayed; the *Non-positional features* panel has a list of annotations of the whole protein and not to a specific region of it such as publications; and the *filtering* panel makes use of the DAS ontologies to create hierarchical filters(displayed as trees, and referred to as *Filtering Trees* in the rest of the document) for the annotations through their category, type or server [24].

In contrast to Spice, Dasty2 does not allow the user to query from both protein sequence and structure; its starting point is always a protein sequence accession number. It is possible, however, to see the related structures of the target protein and, in the case of having more than one structure, the user can choose one to visualize (see the *Protein structure* panel in Figure2.5).

According to *Jimenez, et al*[24], Dasty2 *"facilitates the interaction between the user and the information stored in DAS servers"* and for that it uses the tools that the Web 2.0 provides.

I am listed as one of the authors in this publication because I contributed to the development of some of the panels of Dasty2.

2.3.5 Previous Writeback Implementation

Previous work on this topic has implemented a DAS writeback server as a proof of concept [19] in a Masters thesis at the Chalmers University of Technology. The graphical user interface was built using JSP (Java Server Pages) and the servers are Java servlets, however, there is currently no DAS client able to use this technology. The initial idea was to use this software as a starting point to implement the writeback server with the required capabilities for a Collaborative Annotation System, however, the actual implementation used to store the new annotations was incompatible with the concept of meta-annotation, which is one

the same project or the same kind of content. These groups are called Registry label

of the fundamental ideas of this project. Nonetheless, the experiences and results of that project were very useful and enabled us to avoid several potential issues.

2.4 RESTful web services

RESTful web services implement remote procedure calls across the Web as an alternative solution to SOAP[11] web services. The major strength of the RESTful strategy is probably that it is based on such widely adopted standards as HTTP, XML, URI and MIME[12] , that makes REST and therefore DAS technologies easy to implement and attractive to both developers and final users. This is mainly because, in all likelihood, they already know how to use those technologies. A comparison between SOAP and REST web services can be found in [40].

One of the main features of the REST architecture is to have a *Uniform Interface* which means that all the 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: Create, Read, Update and Delete that correspond to the HTTP methods PUT, GET, POST and DELETE. Those operations *"are broadly applicable but they also help uphold specific Web architectural properties"* [48].

The idea of specifying operations for publishing and editing resources using HTTP is not novel; AtomPub is a proposed protocol for publishing and editing Web Resources using HTTP [18]. Google also has defined a protocol based on Atom, AtomPub and RSS2.0 [17]. To a large extent, the writeback specification used for this implementation is a combination of features of those protocols, plus the inherent requirements of the DAS technology.

[11] *SOAP*: Simple Object Access Protocol, is a protocol specification for exchanging structured information in the implementation of Web Services in computer networks.

[12] *MIME*: Multipurpose Internet Mail Extensions is an Internet standard that extends the format of e-mail

2.5 User Centred Design

The main goal of the User Centred Design is to involve the user as the focus for development. It looks to create applications closer to the needs of the final operator of the system, aiming to make the processes easier and more understandable.

Usability testing is an important aspect of any methodology that puts the user, rather than the application at the center of the development process. The concept of usability was specified in the norm ISO 9241 part 11 and is described as the *"Extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use"* [22].

A usability test might be executed as a *Formative Evaluation* of the user Interface. This means that its goal is to help improve the interfaces; thinking aloud protocols, constructive interaction and heuristic evaluation are some examples of methods that aim to reach this goal [37]. There are other methods that assist in the process of designing the interface, for instance, the use of *paper prototyping* involves the user in early stages of the design, where the interface can be as simple as hand-sketched drafts of the windows, menus, dialog boxes, pages, popup messages, etc. and therefore changes are as easy as redrawing some of the sketches.

Several kinds of tests can be implemented which aim to identify usability issues, critical errors or suggestions for improvement from users. All these methods are dependent on the skills, number and interest of the participant users.

A Heuristic Evaluation is a method created to use the skills of a small set of usability experts. The method consists of group and analise each of the users' opinions about how good or bad the interface is with respect to a set of criteria: heuristics[13]. The heuristics include aspects like the simplicity of the language, the consistency and clarity of the elements, user control, the error handling and the efficiency of use [38]. As Nielson describes in [37], the level of expertise of the subjects influences the results of the experiment, and therefore the selection of the test subjects plays a determining part in the success of it.

Think-Aloud protocols have been used to test user-computer interaction since the early 1980s. The authors of [10] made a distinction between concurrent and retrospective think-

[13]The number of criteria to use has been defined between 9 and 14 depending on the author

aloud methods. In the first one, the subject of the test has to accomplish a set of tasks using the software whilst describing what he is doing, what he is expecting, problems found or suggestions. The retrospective Think-Aloud method consists of recording the user executing a set of tasks in silence, and afterwards showing the video to the user while he describes what he was doing at every moment.

An alternative derived from Think-Aloud protocols called Constructive Interaction was originally developed by Miyake [34], looking to create a framework to study the iterative nature of understanding, and it was tested in the process of understanding a sewing machine. This method was applied for the first time in a user-computer interaction environment for O'Malley *et al.* [39]. In such a case the study was about the Unix C-shell(Command interpreter) and the rules of the pass of variable values to subordinated processes.

Basically, Constructive Interaction consists of executing the tasks in dyads, one of the users is the actor (who has the control of the computer) and the other is the co-actor. The instructions for the test subjects indicate that they consult each other before any action and avoid contact with the facilitator. In this way the ideas are expressed in a more natural way, as a normal communication between the parts of the dyad. Some authors claim that this method required half of the number of experiments to detect the same number of problems, which is a clear advantage in terms of time to acquire that knowledge. This is the case in a comparison of methods using children as users to test a new model of mobile phone [5], however, the comparison in [10] using a University Library System contradicts that statement.

2.6 Key points

The content of this chapter was fundamental for the proper execution of the whole project. It cements the basis for each part of the development process of the Collaborative Annotation System. Here is a list of key points extracted from this chapter that played an important role in different stages of the project.

1. The formal model for annotations introduced by Agosti [3] was the inspiration to define some of the characteristics of our system, for instance, to keep the historical

changes of an annotation (Section 3.2.2) and to display a view of the versions of the annotation (Figure 5.5(d)).

2. Projects like Annotea [27] that take advantage of the correlation between web technologies and annotations, plus all the Web 2.0 sites that use comments, tags and other types of annotations, influenced the decision to use a web-based DAS client over the stand alone options (Section 5.3).

3. The decision points and suggestions about annotations defined by Gazan [16] and Marshal [32] were considered at the moment of defining the principles and strategies of the Collaborative Annotation System (Section 3.3).

4. The experiences of projects like wiki-proteins [36] and gene-wiki [49] teach us the importance of the trust that the biological community should have in the system to make it really valuable. Another important outcome from those projects is the strategy of starting the community from a well recognized knowledge base: UMLS, UniProtKB, IntAct and Gene Ontology[14] in the case of wikiproteins; Entrez, Gene Ontology and the PDB for Gene Wiki; and all the data sources of the Distributed Annotation System for our Collaborative Annotation System.

5. A full understanding of the architectural behavior of the Distributed Annotation System was crucial to be able to propose the extension described in the Section 3.4.

6. The proposed extension(Section 4.4.2) to the DAS protocol is mainly inspired in the concepts of RESTful discussed here.

7. The analysis of the servers done in the Section 4.3 was only possible because of the study of the server's implementation in this chapter (Section 2.3.3).

8. The report on DAS clients in Section 2.3.4 was the raw material for the analysis in Section 5.3.

9. User Centred Design was the main component of the approach followed during the whole project. The concepts in Section 2.5 were relevant in all the stages, but have special importance in Chapter Chapter 6.

[14] *Gene Ontology*: The Gene Ontology project is a major bioinformatics initiative with the aim of standardizing the representation of gene and gene product attributes across species and databases.

Chapter 3

Writeback Protocol and Architecture

3.1 Introduction

The current DAS architecture (Section 2.3.1) does not include any component responsible for dealing with feedback from the user.

This Chapter presents a proposal defining "where" the feedback information is going to be managed (Architecture) and "how" this new component is going to communicate with other DAS components (Protocol).

We propose the DAS writeback server as a third-party component in the DAS architecture to store and manage the information that comes from the users.

There was a set of communication rules between a DAS client and a writeback server following the DAS 2.0 specification, this protocol was declared deprecated during the development of this project, therefore another specification was necessary. We have proposed an extension for the DAS protocol that includes the writeback capabilities following version 1.53 of the specification.

This chapter is organized in the following way: the definition of the problem; a list of the principles and strategies followed during the execution of this project; the proposed architectural extensions to DAS to support a writeback capability; a description of both protocols; and finally some conclusions and lessons learnt at this stage.

3.2 Problem Definition

The Distributed Annotation Systems have delivered an integration and interoperability layer over heterogeneous sources of information. However, so far no protocol has been adopted so that user-based contributions can augment the stored information.

An extension of the protocol is required for any writeback activity, such a specification has to consider the architectural behaviour of DAS to add a server that stores the information and interacts with the user requests establishing the communication rules.

3.3 Principles and Strategies

A DAS writeback server should have, at the very least, the methods for basic reading/writing operations. In database theory, this is known as CRUD (Create, Read, Update and Delete). The implementation of the writing operations in the DAS sources creates a conflict of interest because the owners of the information are willing to share it through DAS, but do not want regular users to change their data without any kind of curational process.

The approach of this project is to deal with this issue, and it consists of the following principles and the application of strategies to achieve them:

1. *The information of the DAS sources should be protected*: The fact that some institutions give free access to query their annotations does not imply that they want external manipulation of their data. Most of them have strong policies about how to, and who can, upload new information into their databases. The next set of strategies tries to provide such protection to the sources:
    - The original data will never be changed by a user of this system.
    - The information in the writeback server should be about additions, modifications or deletions of features in current data sources.
    - The writeback information is optional for DAS; it is the client (as a predefined behaviour), or in the best-case scenario, the user (during runtime) who chooses whether or not to use the collaborative information that is created through the writeback server.

2. *The system should be trusted by the user*: The success of a collaborative environment is strongly correlated with the level of trust its users have in it. It is consequently important to provide the user with information about the source of data and tools to make use of it. These are the strategies to follow in order to reach this objective.

    - The writeback server should store (and provide by request) the historical changes that any annotation has experienced.
    - The writeback server should have a method to identify the users.
    - The writeback server should be able to provide meta-information about the annotation, such as the author, date, etc.

3. *The system should promote interaction between the server and users*: The use of writeback capabilities should be intuitive; the user should feel that retrieving, adding, editing or deleting annotations are natural tasks of the system. Bearing these principles in mind, the following strategies have been defined:

    - The writeback information should be accessed in the same way as any other DAS server.
    - The writeback client should be as similar to a current client as possible, or preferably even extend current clients and implement the writeback functions following the design patterns of the client that is being extended.
    - The protocol details and technology issues should be hidden from the user.

3.4 Proposed architecture

In order to put in place the first principle, a simple idea is to take into account the following: *"The writeback server should be a third party server that stores the changes to the data set"*. In addition, since it is saving changes to annotations, and those changes can be seen as annotations themselves, the server will provide methods to annotate[1] annotations, or meta-annotations as they will be referred to in the rest of this document.

Figure 3.1 shows where in time the writeback server is in the DAS architecture. Firstly, it is necessary to highlight that, for reading purposes, the writeback server behaves as

[1] Three kind of annotations: Create, Update and Delete

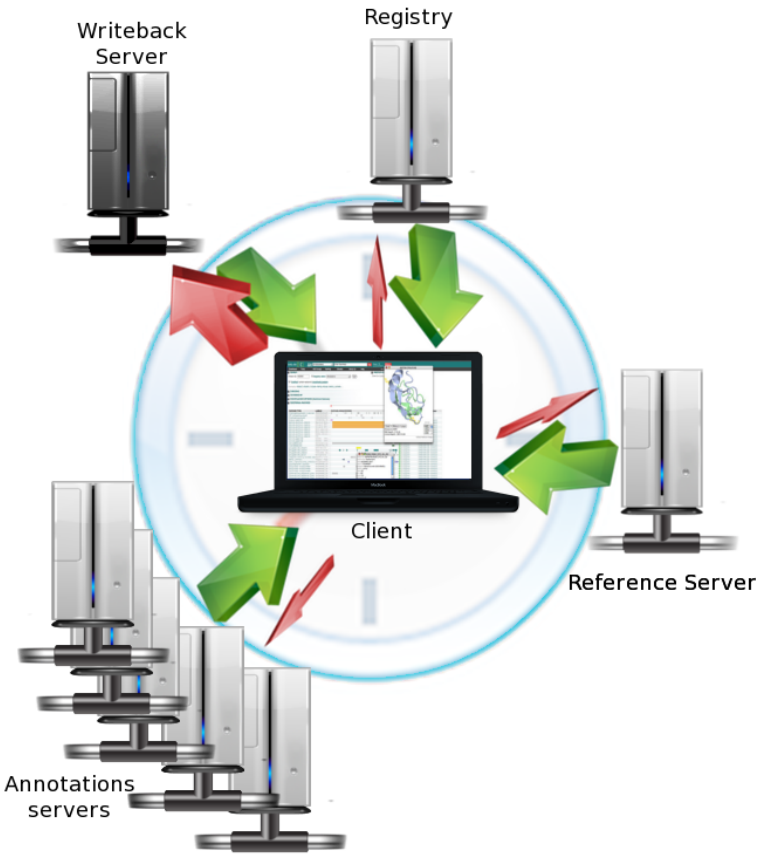

Figure 3.1: *Writeback in the DAS Architecture*: Extension of the DAS architecture in the Figure 2.1. A third-party writeback server is the last step that the client queries, and its response is used to update the information of the annotation servers. The communication with the writeback server has the peculiarity that the amount of information sent by the client is considerably bigger than for any other server

another annotation server; it is just the last one in the queue and the way this information is rendered is the responsibility of the client.

The most interesting things from the server's point of view are the requests related to feedback. For those cases, the main difference is the amount of information in the request (shown in Figure 3.1 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. Therefore, the communication with the writeback server is extended beyond the display of the graphic that compiles the information of all the servers. This is when the user starts to interact with the information, transforming the client from a pure visualization tool to an interactive

interface between the user and the DAS data knowledge.

Another important thing to remark on is the time; in both Figures 2.1 and 3.1, there is a clock in the background, which represents the chronological order of how the actions happen. From this, it can be inferred that the interaction between client and writeback happens after the client has conglomerated or even displayed all the information for the target protein. This is because it is only then that the user has a complete landscape view to take the decision to add, update or delete a feature.

3.5 Writeback Protocol

The writeback was mentioned for the first time in a specification in the DAS2.0 document. DAS1.53 does not include this feature. Initially, our idea was to use the DAS2.0 specification, but given the low adoption of it, an opportunity arose to define a specification that was closer to DAS1.53. However, the release of a beta version of DAS 1.6 was in March 2009, and by that time the progress of an implementation for the DAS2.0 writeback protocol was advanced. Here we presente the two protocols used.

3.5.1 Writeback based on DAS 2.0

Although DAS2.0 followed the same goals of DAS1.53 about the sharing of information through a federated system, the documents defined and method proposed had dramatic changes in comparison with the current DAS version 1.53.

DAS 2.0 was an ambitious project that makes extensive use of Web technologies such as URIs, HTTP, XML, and REST among others. The DAS2.0 specification is not backward compatible with the DAS1.x versions, and therefore all the documents defined in the protocol are different to any previous version.

Those differences are considerably important for this project because none of the studied DAS servers in Section 4.3 uses DAS2.0 as its protocol. Nonetheless, DAS2.0 was the first document that includes a writeback specification and because of that, an effort to conciliate the protocols was planned:

- The output format will be based on DAS1.53 as most of the clients support this version.
- The logic model of the server for DAS1.53 will be reused, especially for output purposes.
- The input format for the writeback functionality will follow the DAS2.0 specification.
- The new commands of the writeback server will be *writeback* (for create, update and delete a feature) and *historical*.
- A logic model for the entities of DAS2.0 will be required.
- The database will be designed according to the DAS2.0 specification.
- A translator will be required between the logic models.

The writeback specification included in DAS 2.0 is published in biodas.org[2].

Basically, this document defines a writeback document, which includes the information that the user wants to submit to the server, including the operation to execute with it(Create, Update or Delete). It also defines some strategies about how to manage the feature IDs as URIs, how the server should respond, and how to deal with errors. Below is an example of a feature document in DAS2.0.

```
<?xml version="1.0" standalone='no'?>
<FEATURES   xmlns="http://biodas.org/documents/das2"
      xml:base="http://www.example.org/volvox/1/">
  <FEATURE  uri="feature/hit12"
        type="type/est-alignment"
        created="2001-12-15T22:43:36"
        modified="2004-09-26T21:10:15" >
    <LOC segment="segment/Chr3" range="1201:1400:1" />
    <PART uri="feature/hit12.hsp1" />
    <PART uri="feature/hit12.hsp2" />
    <PROP key="est2genomescore" value="180" />
  </FEATURE>
</FEATURES>
```

[2] http://biodas.org/documents/das2/das2_writeback.html 2006

3.5.2 Writeback based in DAS 1.53

The DAS community decided to declare DAS 1.53E as the current specification and DAS 1.6 as the next one, in an effort to implement some of the features of DAS 2.0, but keeping the format and methods as compatible to previous versions as possible.

We consider that if that is the policy to follow for the DAS community, we should propose a writeback specification along the same line of ideas.

The proposed specification can be found on the DAS1.6E web page[3]. This page is the result of an agreement in the 2009 DAS workshop to have a common place for extensions to the current versions.

Basically, the specification proposes that both input and output documents for the writeback should follow the DASGFF format (See the next code example); the HTTP method indicates what to do with the received document (create, update or delete a feature) and the HTTP codes used for DAS are still valid here and will indicate success or failure of the requested command.

The command to execute with that information should be specified in the HTTP method itself, following the principle of *Uniform Interface* of RESTful web services (Section 2.4).

```
1 <?xml version="1.0" standalone='no'?>
2 <!DOCTYPE DASGFF SYSTEM "http://www.biodas.org/dtd/dasgff.dtd">
3 <DASGFF>
4  <GFF version="1.0" href="http://www.ebi.ac.uk/das-srv/uniprot/das/uniprot/
      features?segment=P05067">
5    <SEGMENT id="P05067" start="1" stop="770" version="7
       dd43312cd29a262acdc0517230bc5ca">
6      <FEATURE id="UNIPROTKB_P05067_KEYWORD_Disease" label="Disease
         mutation">
7       <TYPE id="BS:01019" category="inferred by curator (ECO:0000001)">
           disease</TYPE>
8       <METHOD id="UniProt">UniProt</METHOD>
9       <START>10</START>
10      <END>40</END>
11      <SCORE>0.0</SCORE>
12      <ORIENTATION>0</ORIENTATION>
```

[3] http://www.biodas.org/wiki/DAS1.6E#DAS_writeback 2009

```
13        <PHASE>-</PHASE>
14        <LINK href="http://www.uniprot.org/uniprot/P05067">http://www.
              uniprot.org/uniprot/P05067</LINK>
15        <NOTE>Adding a new feature!</NOTE>
16        <NOTE>USER=userlogin</NOTE>
17      </FEATURE>
18    </SEGMENT>
19  </GFF>
20 </DASGFF>
```

3.6 Conclusions and Lessons

An architecture that extends the current DAS system has been proposed in order to give the users the opportunity to provide information for current or new annotations.

Two alternatives for the writeback specification were presented: One was adopted from the DAS2.0 protocol and a second one was created inspired by the current DAS versions.

Although we consider DAS2.0 a valuable effort with great improvements for DAS, its lack of compatibility with previous versions, did not allow it to become the standard protocol, we consider this a valuable lesson for any project: *"The support of previous versions should be mandatory for any new release of any technology"*.

The decision of defining a new protocol was influenced by the DAS community because of the feedback received during the DAS workshop 2009, and through the DAS mail list where the people involved in the new version of the protocol and the creation of applications for DAS show their interest in this project.

The reuse of the DASGFF language plus the HTTP method in the proposed protocol, brings the advantage of being extensible to next version of the DAS specification, because the method will be the same, and the format should be upgraded to the new version.

Chapter 4

DAS writeback server

4.1 Introduction

We present a writeback server for DAS. This server allows the DAS ecosystem to capture annotations provided by users, and retrieve this information in a DAS oriented way. In order to do this, it was necessary to extend the architecture of DAS (Section 2.3.1) to include a writeback server as described in Section 3.4.

This chapter describes two implementations(following the two specifications) of a server that supports a series of operations allowing for the capture of user feedback and the immediate availability of that feedback to the public. It seeks to create the conditions necessary for a collaborative annotation environment.

Several DAS servers were considered to be extended with the writeback functionalities, a comparative analysis was done and MyDas was selected, therefore both implementations of the protocols are extensions of MyDas.

This chapter is organized in the following way: The definition of the problem; A report of the most important DAS servers with special focus on MyDAS, the chosen system to implement the writeback capabilities; The design and implementation details, including reasons why two different implementations were created; and finally, a brief discussion of the results and a comparison of the implementations.

4.2 Problem Definition

Chapter 3 presents two different specifications of the writeback for DAS: One is introduced as a part of the DAS 2.0 project, and the other one is our proposal based on the current DAS protocol 1.53.

In order to establish the viability of a specification, an implementation of the writeback server has to be done– it helps to capture technical errors of the protocol, but more importantly, it creates a real basis to start building a community for the Collaborative Annotation System.

In summary the problem to solve in this Chapter is to demonstrate that the specifications are implementable, what advantages/disadvantages have one agains the other one, and which one should be chosen to work with for the Collaborative Annotation System.

4.3 Analysis of DAS Servers

There was an initial decision taked whether it is convenient to extend an existing server or to create a new one that supports the writing/reading operations. The reading component is already solved for the DAS protocol and all the current implementations of this specification have those capabilities. This is the reason for the extension option having been chosen. However, it raises another question about which of the current servers should be extended.

Several DAS servers were studied: Dazzle, LDAS, MyDas and Pro-server. A description of these clients can be found in the Section 2.3.3.

Table 4.1 summarizes the main features of these servers.

Several criteria were taken into account in the choice of which server to extend. Initially, LDAS was discarded because it seemed that there has been little development activity on it for more than 5 years. In addition, LDAS aims to serve to predefined environments, such as a MySQL database or a plain GFF file and it does not have the required flexibility. Pro-server has been written in Perl and has probably been the most active project for the last year. The advantages of Pro-server are similar to those of MyDas. I have more experi-

Table 4.1: Comparison of DAS servers

Feature	Dazzle	LDAS	MyDas	Pro-server
Language	Java	Perl	Java	Perl
Last Release	2008	2002	2007	2009
DAS version	1.53E	1.53	1.53	1.60
Physical storage	User def.	MySQL	User def.	User def.
Some Protein Sources		-	UniProt,Pride	Encode,Pfam
Responsible	Sanger Inst.	CSHL	EMBL-EBI	Sanger Inst.
Open Source	YES	YES	YES	YES
Documentation (how to)	Good	Good	Good	Good
Documentation (software)	Good	Poor	Good	Good

ence in Java than in Perl projects, therefore the decision was between Dazzle and MyDAS, and, although the software architectural patterns are similar in both projects, the package organization and the way the Model-View-Control (MVC) pattern was implemented influenced the decision to use MyDas. This concept is useful for the purposes of this project because it allows for the extension of MyDas to support the writeback capabilities whilst at the same time, it allows for the independent implementation of the data source with the necessary features for the Collaborative Annotation System.

4.4 Implementation Details

We implemented two different servers; the first one follows the DAS2.0 specification(Section 3.5.1) and the second one is based on a proposed extension of the protocol(Section 3.5.2). Both implementations are extensions of MyDas. Below are the details of both implementations, followed by a discussion of the pros and cons of each one.

4.4.1 Writeback for DAS 2.0

The release of the DAS2.0 protocol preceded the start of our project by almost one year. Nevertheless, even then there was no stable implementation of this protocol. Therefore, as a first attempt to create a DAS server with read/writing capabilities, the writeback

specification in DAS2.0[1] was followed as described below.

The controller of MyDas, which is a Java servlet, was extended to process the commands *writeback* and *historical*, moreover, it now has the capability to capture the writeback document from the POST message. This document is parsed into a new set of classes that expand the MyDas model in order to deal with the syntax of DAS2.0 for writeback documents. Figure 4.1 represents the class diagram of the classes that extend the MyDas model.

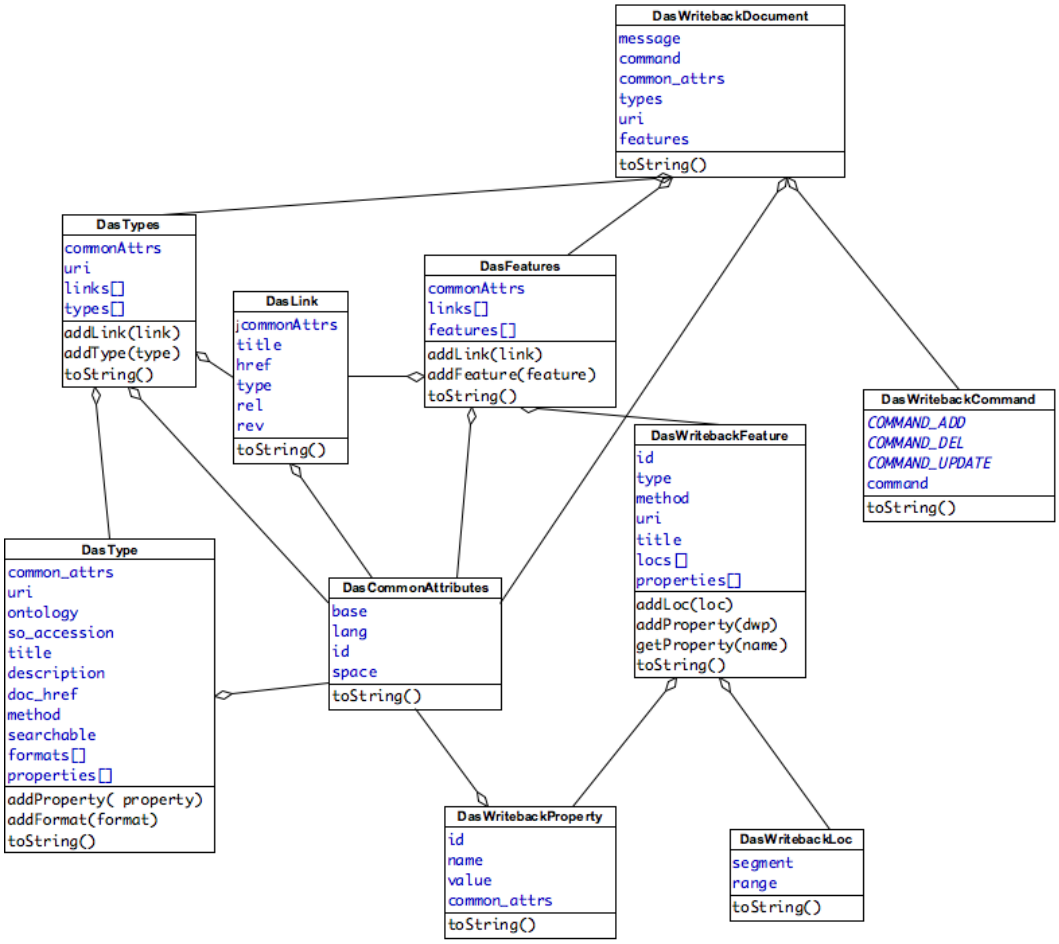

Figure 4.1: Writeback Class Diagram for the model

Additionally, a new Data Source Interface was created in order to define the methods that should be implemented to provide writeback functionalities.

[1] http://biodas.org/documents/das2/das2_writeback.html 2006

The logic used to process the configuration file was also extended in order to accommodate the new interface. A class that implements this Interface should define these methods:

```
1 public void addFeatures (DasWritebackDocument doc)
2 public void deleteFeatures (DasWritebackDocument doc)
3 public Collection<DasAnnotatedSegment> getHistorical(String fetureId)
```

The method *addFeatures()* should receive an instance of the writeback document that contains a set of features to be included in the database. This method is also used for updates, so the logic of this method should check if the feature already exists in order to decide whether to update or add a new annotation.

The method *deleteFeatures()* should receive an instance of the writeback document that contains a set of features that will be deleted in the database.

The method *getHistorical()* receives the ID of a feature and should return all the different versions of that feature in the server.

The basis on which this component was built is the documentation of the DAS2.0 writeback, in particular the formal XML schema for the writeback document[2]. Another important source of information was the result of discussions about the implementation of this protocol via the mailing list of the DAS community, including people who work on the creation of the protocol, as well as current developers of different DAS based components.

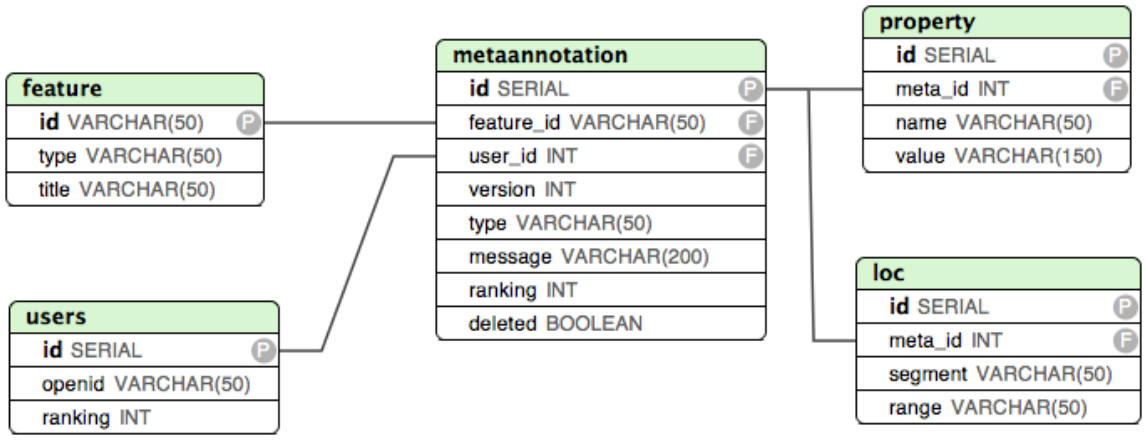

Figure 4.2: Writeback Database Diagram (First implementation)

[2] http://biodas.org/documents/das2/writeback.rnc 2007

Once the framework was ready, the second step was to implement the interface. The new data source should implement the writeback interface, but also one of the MyDas interfaces that was previously defined. This is because the data source should not just have edition/creation functionalities but also behave as a normal DAS source providing the features on the demand of the users.

This specific source stores the information in a PostgresSQL database. Figure 4.2 displays the database diagram.

This database stores a single instance in the table feature for each unique *feature* that is created/uploaded/inserted. The table *metaannotation* has as many entries as user requests received by the writeback server. Additions, updates and deletions are just new entries in the table *metaannotation*, therefore the database will accumulate all the historic changes for each feature and it will be feasible to roll back to a previous state of the annotation.

The feature is stored in a different table because it is possible to have the same feature in more than one location, this is a DAS2.0 addition.

Most of the information in the DAS writeback document is in the PROP tags that provide a generic way to link information with a feature. All this information is saved in the table *property*.

Finally, the users table contains the record of who has created or edited an annotation. Users are identified through an OpenId[3] login. The server should verify this ID against an OpenId server before storing any of the commands from its request.

The source code of this implementation is available through an SVN server in sourceforge[4].

The main difficulty in the implementation of this server was how to reconcile the difference between versions. MyDas is a server for version 1.53 of the protocol and the writeback is a part of the DAS 2.0 specification which requires some kind of translation between the input data (writeback document DAS2.0) and the output data (DASGFF format DAS 1.53). Although this implementation deals with such translations, the mix of versions could potentially create confusion in users.

[3] http://www.openid.org 2009
[4] https://mydaswb.svn.sourceforge.net/svnroot/mydaswb/MyDas_first_WB_version/ 2009

4.4.2 Writeback for DAS 1.53

This implementation of the writeback is an extension of MyDas and is, therefore, based on DAS1.53. It is possible to have implementations for 1.53E or even 1.60 with further revision.

This specification is inspired in the *Uniform Interface* method of RESTful web services, and as a web application the best way to implemented is using the different HTTP methods. Given that MyDas is a Java program and its controller is a Java servlet, the implementation of the HTTP listener for the methods POST, PUT and DELETE, corresponds to the methods *doPost()*, *doPut()* and *doDelete()*. In an effort to invade the MyDas code as little as possible, the only function of the override methods is to redirect to the correspondent method in a new class called MyDasWriteback that behaves as the controller for the writeback functionalities. This class uses the DasParser to create a WritebackDocument instance that contains all the information of the request in an object oriented structure.

As in the previous implementation, a new data source interface was created but in this case, the methods to implement should be:

```
1 public DasAnnotatedSegment create (WritebackDocument document)
2 public DasAnnotatedSegment update (WritebackDocument document, String id)
3 public DasAnnotatedSegment delete (String featureId, String userId, String
      segmentId)
4 public Collection<DasAnnotatedSegment> getHistorical (String featureId)
```

After the user is authenticated, the controller looks for the writeback data source implementation in the configuration file. For the Collaborative Annotation System, the data source was created to store the information in a PostgreSQL database that follows the relationship diagram shown in Figure 4.3. This design tries to bring an approximation of the structure of the DASGFF format[5] that follows the basic rules of normalization for database design, incurring as little redundancy as possible. For example, many features can be related with the type X, but X will only occur once in the table *type*. The basic idea of the design was that for each element in the specification, there is a table in this database, and for any parameter there is a field. New features are stored using the table *metaannotation* and the link tables. If the feature contains a type that is not in the

[5] As its defined in the DAS1.53 specification

database, this will be added. This idea of incremental updates for types applies to the rest of the tables as well.

The table *metaannotation* stores all the actions that have been applied to a specific feature. It also contains meta-information such as the version, the date and a ranking, which has not been used in this version, but it will be important to define a ranking system that allows users to choose the most trusted features.

Both methods *create(...)* and *update(...)* behave in a very similar way, adding information to the table *metaannotation* as explained before. Their only difference is the assignment of the ID, because if it is an addition then a new ID is created, but if is an update then the ID is the URI formed by the concatenation of the server URL and the feature ID in the XML. It also verifies whether an ID has been used before, in which case it will keep the same ID. The version of this feature, therefore, is incremental.

When a feature is created or edited for the first time, the parent segment is also added to the database, or recovery from it if is already there. One of the attributes of segment is the version, in the case that a sequence changes, the version of the segment is different, this helps to identify to which version a feature is annotating.

The method *delete(...)* does not require a writeback document; the IDs of feature, segment and user is enough to create a meta-annotation that informs a client that a specific feature has been tagged as deleted.

The method *getHistorical()* displays all the versions of the meta-annotations for a specific feature ID.

The writeback data source also implements the AnnotationDataSource Interface in order to provide the responses for the regular DAS commands (feature, types, etc.).

The fact that the design is so close to the DASGFF implies that the database is sensitive to the version of the protocol. For instance, some fields that are part of the 1.53E version do not appear in this database, therefore, in most of the cases, a new version of the protocol will imply a new version of the database and consequently a new version of the Data Source. However, the extensions to the controller of MyDas won't require any updates.

In summary this extension can be divided in four components: (1) The modifications inside

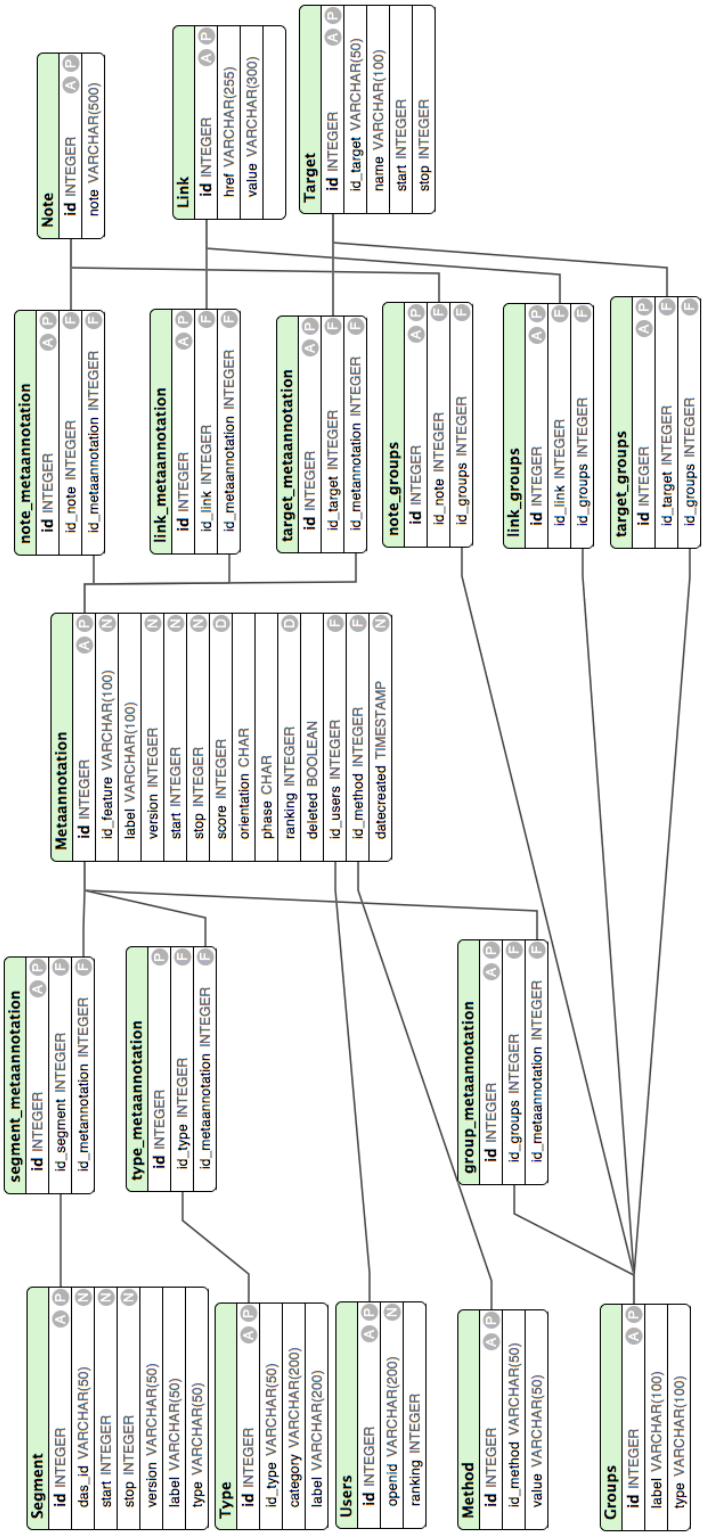

Figure 4.3: Writeback Database Diagram (Second implementation)

the MyDas code to recognize the new commands, and the definition of an interface for a writeback data source, (2) The model extension to allow the access to parameters that were private and a class WritebackDocument to encapsulate instances of the model, (3) The writeback controller, which receive the request from the core, parse the message into the new Model and invoke the right Data Source that implements the Writeback Interface, and finally, (4) An implementation of the writeback interface, with a Database Manager using JDBC for the PostgreSQL RDBMS.

All this development was made in Java SE 6[6] using Eclipse Europa[7] as IDE and Apache Tomcat Version 6.0.18[8] as a servlet Server

4.4.3 Discussion

As seen previously, two different implementations were created, one is based on the DAS2.0 protocol and the other one provides a new specification based on version 1.53 of DAS. Both approaches have pros and cons, and here we present some differences:

New Format (DAS2.0) vs. reuse of DASGFF (DAS1.53) The first implementation uses a new communication format defined in the DAS2.0 specification (See the code example in Section 4.4.1); the second implementation reuses the DASGFF format (See the code example in Section 4.4.2). The creation of a new format has the advantage of being more expressive by providing the exact element that represents the piece of information that the message requires. For example, the possibility of using the element *LOC* more than once in DAS2.0 provides the opportunity to use the same annotation in more than one segment. However, the benefit of expressiveness has been sacrificed in DAS2.0 in order to get more generalization. This is evident given that a large part of the information about a feature is contained in the *PROP* element, which is a general use tag for any property of the feature. On the other hand, the reuse of DASGFF format ensures that all the information provided by a DAS server is included in the message, and any other feature inherent in the write-back can be included using the element *NOTE* in a similar way to *PROP*. The fact

[6] http://java.sun.com/javase/ 2009
[7] http://www.eclipse.org/europa/ 2007
[8] http://tomcat.apache.org/ 2010

that DAS developers are already familiar with the DASGFF format is an advantage, as they won't require much effort to understand the few new features and, therefore, to understand the whole protocol.

New model for writeback vs. Reuse of MyDas model This is an implementation difference because the class model is the representation of the writeback document in an object-oriented structure. In the first implementation, a whole model was defined (Figure 4.1), for the second one, the model of the server implementation was reused, but it was necessary to extend the current classes by adding writing methods to the private parameters. The real difficulty in the first approach was the need for a translator between models, because the server is based on DAS1.53, the output requires that the information is contained in that model. Even if most of the cases have the same information, there are examples of incompatibilities, such as the *LOC* element explained above. This issue would not exist if the server was based on DAS2.0; however, that protocol was not adopted, as explained above. This translation problem is not exclusive to the writeback, the developers of DAS2.0 were trying to provide a solution for it.[9]

Embedded functions vs. Uniform Interface In the first specification the function requested to the server should be indicated as part of the writeback document, while in the proposed specification the function corresponds to the HTTP methods used to communicate with the server. This is a similar debate to the one in [40], in that it refers to the Uniform Interface. In the case of DAS2.0, as in the case of 'Big' web services, the actions are embedded in the document. The main advantage of this approach is that the format becomes generic for a virtually unlimited set of methods. However, Vinoski maintains in [48], that this feature is ironically inferring that the services are not as reusable as they should be, because the freedom to create any method arises with the creation of contracts and descriptions of the services; those are becoming so specific that *"the likelihood that an interface will fit what a client application requires shrinks as the interface's specificity increases"*. RESTFul, in contrast, proposes the use of a Uniform interface, which, in the web, is the use of the methods GET, POST, PUT and DELETE. Within the scope of a writeback for DAS, those methods are reasonable enough to reach the manipulation goals.

Small database size vs. Medium database size The Designed Database for the first

[9] http://lists.open-bio.org/pipermail/das2/2008-October/001055.html 2008

implementation (Figure 4.2) has 5 tables, which, when compared with the 18 tables of the second implementation (Figure 4.3), looks like a very small number. However,both of them are capable of storing the meta-annotation information. An advantage of the first model is that, given the generality of the table property, this database will probably not change in the case of a new version of the protocol (The changes will be in the model and the data source). The second version is more susceptible to changes in the protocol, however, its similarity to the model simplifies all the operations of the object-relational mapping.

4.5 Conclusions and Lessons

Several DAS server were considered as candidates to be extended with writeback capabilities, and finally MyDas was chosen, mainly for its architecture and its experimentally proven robustness.

Two different writeback servers were developed; one using the specification of the DAS2.0 protocol, and for the second one a protocol specification was proposed to the DAS community and the implementation was created to be compatible with the version 1.53 of the protocol.

In general terms, the second implementation was easier and faster to develop because of the simplicity of dealing with just one protocol, but also because of the experience and clarity about the problem gained whilst implementing the first version.

The main benefit of this is that although the scope of this project is limited to proteins as a coordinate system, this writeback server is capable of managing other coordinate systems such as the ones for genomic or structural information.

Another advantage of the second implementation is that, given that it reuses the DASGFF format, it is potentially easier to extend DAS clients because they are already in that format.

For these reasons, the implementation of the writeback using the proposed specification was the selected one in this project to reach the goal of providing a collaborative annotation environment for the DAS system. Therefore, hereafter, every time the writeback server is

mentioned it will refer to the second implementation.

We propose to have one official writeback server per coordinate system, this will facilitate the implementation of the writeback clients, because usually a client does not query more than one coordinate system at a time. The information of the official writeback for a coordinate system can be maintained in the DAS registry.

Unofficial writeback installations can be useful for small groups in annotation projects, however these won't be held in the registry, and therefore, public clients won't use this information.

Chapter 5

DAS writeback client

5.1 Introduction

As a federated system, the Distributed Annotation System, DAS, delegates most of the integration responsibilities to its clients, giving it the architecture of a *"dumb server, clever client"* [23]. As a consequence, if the goal is to capture feedback from the users (*Writeback*), the client should be able to execute several tasks related to both logic and user interaction.

One of the goals of this project is to create the perception for users that the writeback functions in a client are native and can be used in a natural way on the current clients. For this reason, the extension of a current client is preferable to implementing a new client from cratch. The Writeback server behaves as any other DAS server for reading purposes, so potentially, many software routines of an existing client could be reused for the writeback visualization.

With the writeback server described previously as a starting point, this document describes how a client has been extended in order to make the writeback functionalities available in an intuitive and user friendly way.

It is desirable for most of the implemented DAS clients (if not all of them) to extend with writeback capabilities to involve as many users as possible in the collaborative process. For the purposes of this project, however, just one client has been extended: *Dasty2*. Despite

this, the process followed in this project can be applied to other clients.

This chapter starts by defining the specific problem that it covers, followed by a description of DAS clients, considering the most commonly used ones, and then provides a special focus on Dasty2 given that it is the client chosen to be extented. It then gives the proposed solution. The last section has the details of the design and implementation of the solution.

5.2 Problem definition

DAS clients only allow a read-only view of biological annotations. However, it would be useful to enable researchers to create and optionally share new protein annotations as they make new discoveries, both in their research and while viewing other annotations. DAS clients currently do not provide a Graphical User Interface to add, edit, and store the biological annotations. New data or modifications to the annotation in an existing DAS source requires expert access to the original database or structured data. This hampers the possibility of domain experts freely creating and manipulating annotations.

The distributed concept of DAS makes it an appropriate environment to support a *Collaborative Annotation System*, i.e., a system to enhance cooperation among domain experts to enable work on annotation either simultaneously or asynchronously.

Chapter 4 presents a DAS server with writeback capabilities, which is an important step in looking for a Collaborative Annotation System, however the real potential of the system cannot be reached without a client that provides easy and understandable access to the writeback features. Therefore, the problem to solve in this section can be summarized as *"How can the capabilities of a writeback server be implemented into a DAS client?"*

5.3 DAS clients Analysis

Section 2.3.4 shows the importance of a client in the DAS architecture and describes the main characteristics of the most representative protein DAS clients.

In Table 5.1 the main features of the protein DAS clients have been summarized. From

there it is possible to see that the development of DAS clients is a very active field, three of the compared clients have released versions in the current year (2009). It is also evident that there is support for open software for the DAS community since all of the clients are open source. This is also a tendency for clients for other coordinate systems, and even for the DAS servers.

Table 5.1: Comparison of protein DAS clients

Feature	Spice	PFAM	DASher	Dasty2
Language	Java	Javascript	Java	Javascript
Proxy Language	N/A	Perl	N/A	Perl or Php
Last Release	Jan 2008	Mar 2009	Feb 2009	Aug 2009
DAS Version	1.53E	1.53E	1.53E	1.53E
Type	Stand-alone	Web	Stand-alone	Web
Responsible Entity	Sanger Inst.	Sanger Inst.	SBC	EBI, NBN
Open Source	YES	YES	YES	YES
Documentation (how to)	Good	Good	Good	Good
Documentation (software)	Acceptable	Poor	Poor	Poor
3D Alignment	Yes	No	No	Yes
Protein Domain Render	No	Yes	No	No
Line Plots	No	No	Yes	No
Ontology Filter	No	No	No	Yes
Graphic Manipulation	High	Basic	High	High

In deciding which client to extend with the writeback capabilities, several factors were taken into account. Firstly, a collaborative environment as proposed in this document requires the participation of as many users as possible and the web has become the perfect environment for such projects. Considering this, the web based DAS clients fulfill more of the hopes of creating a community for the annotation of proteins. It is proper to mention the valuable effort that Stand-alone protein clients make to provide access to the tools from the web through Java Web Start, however this method requires that Java is pre-installed in the client and the scope of the application runs outside of the browser in an arguably richer interface.

The criterion to select between the two web-based clients was the adaptability of the application in other projects. PFAM offers very specific features that enrich the scope of annotations about domain families, making it proper for the PFAM project purposes, however it has not been used in a different context. Dasty2, in contrast, is being used by UniProt to display non-UniProt annotations. Biosapiens, a European network for genome

annotation also uses Dasty2, as do Spice and ENSEMBL.

For those reasons Dasty2 is the chosen DAS client for the addition of writeback functionality. Moreover Dasty2 offers other features which makes it the perfect candidate for the proposed extensions. As discussed in Section 2.3.4, Dasty2 has a modular structure based on panels so it provides the opportunity to group the writeback features in a new panel, thus isolating the writeback content for those who prefer not to use this information. This isolation, however, is enough to maintain visual relations between those functionalities and the ones belonging natively to Dasty2.

The fact that Dasty2 uses extensibly the technologies related to Web2.0 is very convenient for the development of the writeback capabilities, because the user will interact with the annotations in the same way that they would use a stand-alone application, allowing for communication with the writeback server without loosing the context of the existing annotations.

5.4 Solution Proposed

We consider that a writeback DAS client should implement the following set of characteristics in order to provide an adequate environment for a Collaborative Annotation System:

1. The DAS client should be able to retrieve the information from the writeback server after it has compiled the information from all the other servers.
2. The writeback information should be optional. The user should have a way of choosing whether or not to visualize the writeback annotations.
3. The visualization of the writeback features can be done as a separate set of tracks[1] or overwriting the original information of the sources.
4. Creation, edition and deletion of annotations should be done in the context of the existing annotations.

[1] *Track*: In the context of a graphic of annotations a track is a row where a group of positional features is drawn with the sequence as a reference, in some cases the sequence is the first track of the graphic.

5. The DAS client should provide a method to authenticate the user with the writeback server.

6. The interface should allow for the visualization of the history of changes for a particular annotation in order to give the users an "undo" option.

7. The interface should provide a way to roll back to a previous version of an annotation.

8. Ontologies for the evidence code and the type of annotation should be used in order to standardize the annotations.

9. Validation of the fields of an annotation should be done before sending the message to the server.

Having these requirements in mind, we have created an extended version of Dasty2, which now permits the proposed interaction with the writeback server. The next section contains the details of the design and implementation of this development.

5.5 Design and Implementation Details

Dasty2 is an Ajax application that requires the retrieval of information from external servers, however, most of the browsers impose a restriction that does not allow XMLHttpRequests to be made to any server except the server where your web page came from [30]. One of the strategies to avoid this issue is to have a server that behaves as a proxy between the client running in the browser and any server in the Internet.

Figure 5.1: Communication between Dasty2 and the WriteBack

Figure 5.1 illustrates the component's interaction in a communication between the writeback client and server. When the client requires some information from a remote server, a

request is sent to the proxy. Subsequently, the proxy requests the resource from the server. When the proxy receives a response, it is replicated to the client. The clouds in the graphic representing the Internet indicate that there is not a direct connection between the entities, but they communicate with each other through the Internet. In the case of Dasty2, all response processing is the responsibility of the client and other proxy implementations pre-process the response, so minimizing the client's number of tasks.

This description applies to the communication between Dasty2 and any DAS server (registry, reference or sources); however, for the writeback case there are a couple of differences. Firstly, in the proposed specification of the writeback[2] it is explained that the communication with the server should be using the different HTTP methods (*GET, POST, PUT and DELETE*) according to its function (*query, update, create and delete*). For this reason, the Dasty2 proxy was extended in order to use the correct method.

The second difference is the amount of information; before the writeback, all the requests in Dasty2 were using the *GET* method and, therefore, the information sent from the client to the proxy was as large as 256 characters maximum, which is the URL size limit for most web browsers and servers. With the writeback functionalities, however, the client now requires that an XML document is sent that will probably be bigger than the imposed limit of the URL size, making the use of other methods mandatory and even more appropriate if those new methods coincide with the recommendations of the RESTful standard.

The client side of Dasty2 has been written in JavaScript. Although this language is not considered as robust as other languages like C++ or Java, it allows for the use of the basic object oriented principles such as encapsulation or inheritance [8]. Unfortunately, Dasty2 is not object oriented, but it is organized as a set of function libraries where each library is a file that groups all the functions of a specific module. Despite this, an Object Oriented approach was used for the extension. Figure 5.3 displays the class diagram of the writeback extension for Dasty2.

Figure 5.2 summarizes the interaction between the submodules of the writeback client implementation. The communication between writeback client and server is done using the DAS GFF XML format, which is defined in the DAS specification. The client has to have a logical model (bottom-right, Figure 5.2) to map the DAS GFF format when it is reading from the writeback and it must also start from that model to build the

[2] http://www.biodas.org/wiki/DAS1.6E#DAS_writeback 2009

Figure 5.2: Architecture Diagram for the writeback extension in Dasty2

XML when some information is being sent to the server. The classes coloured in blue in Figure 5.3 correspond to the defined logical model. Currently, these classes have been used solely for writeback purposes, but they can be used to do a re-factoring of Dasty2. Some of the functionalities of Dasty2, such as the drawing of features in the graphic have been replicated in the corresponding class in order to gradually move Dasty2 to an object oriented software. However, because we still use the core of Dasty2 some of the writeback features were replicated inside the Dasty2 libraries (bottom-left, Figure 5.2).

The class *DASParser* is in charge of parsing the XML from the writeback server response for both reading and writing purposes. All this logic uses the Document Object Model DOM to read the file and from there it creates an instance of the model. This class corresponds to the module Parser in the Figure 5.2.

The *WritebackManager* is probably the most important class; in Figure 5.2 it is appreciable that this module is in charge of the orchestration of all the other modules. By the time that Dasty2 has finished loading the DAS servers, a global instance of this class is created,

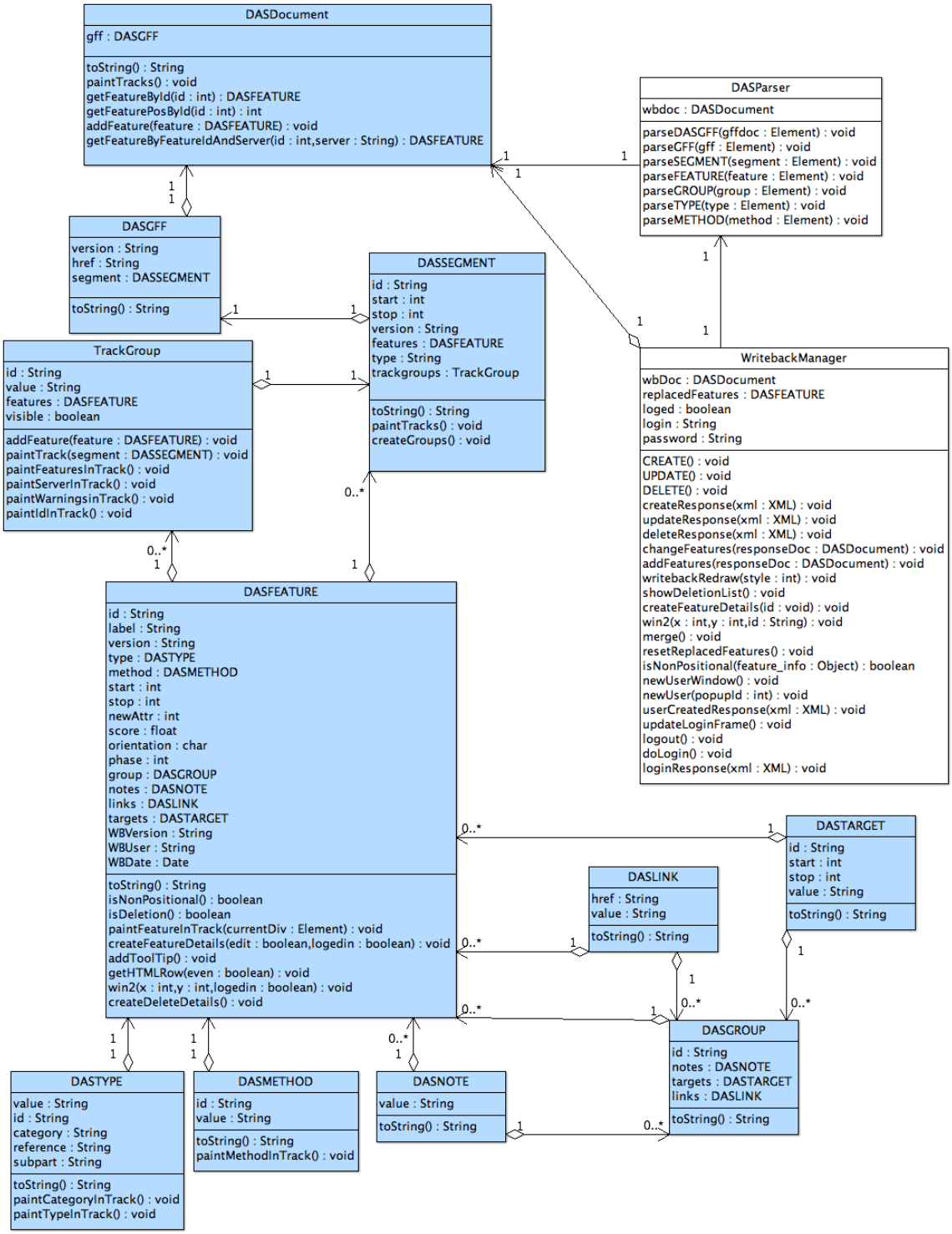

Figure 5.3: Class Diagram of the writeback extension for Dasty2

and then it requests the features that the writeback server has for the protein target. From that moment onwards, this object is the interface between any browser's event related to the writeback and the logic to handle it.

Figure 5.4 is a snapshot of the panel added to Dasty2 for the writeback functions.

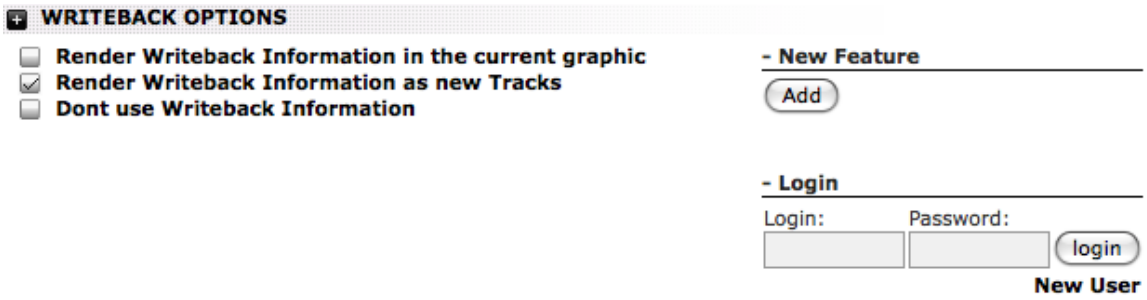

Figure 5.4: Writeback Panel in Dasty2

Next are some details of the functions that have been added to Dasty2 for reading and writing purposes.

5.5.1 Authentication

A small module to allow for user authentication through a login and password was added in the writeback panel (bottom-right corner of Figure 5.4). Any writing function is conditional on a previous login and password validation. The reading functionality does not require authentication.

The module was implemented as a component of the Data Source in MyDas, with a table in the PostgreSQL database that contains the login of the user and the MD5[3] encoded password.

The login and password have to be sent as a part of the message DASGFF through the NOTE element, in the way [KEY]=[VALUE], for example:

```
1 <NOTE>USER=login</NOTE>
2 <NOTE>PASSWORD=keypass</NOTE>
```

[3] *MD5*: Message-Digest algorithm 5

In a preliminary version of the server the authentication was done using OpenID, a third party authentication system. In the isolated tests of the server, that strategy worked well; however when authentication tests were done using Dasty2 it was evident that such a method created several complications in the system. The use of an OpenID server added an extra communication layer in the system, i.e. another server would be required at the very right of Figure 5.1. The authentication should be done for the writeback server and the client should just be the interface for it. The problem is that OpenID works using HTTP location headers with a callback URL; this URL should be pointing to the writeback server in order to execute the action after the authentication is done, and that displays the raw answer in the browser, discarding Dasty2 from the current scope. Alternatively, the callback URL could point to the client, but in this case the client would need to resend the request to the server and extra validations would be required in order to determine if the request has already been authenticated or not. The biggest issue for this alternative is that the server becomes client-dependent, and the goal of using the same writeback server for different kinds of clients becomes unreachable.

Moreover, even if those issues were solved, the fact that the communication for OpenId is by redirecting to their validation web, it becomes an issue for stand-alone clients, forcing them to implement a mini-browser or to look for alternatives to circumvent the issue. This can drastically affect the adoption of the writeback server as the *de facto* standard for the DAS community.

5.5.2 Reading from the writeback

The writeback server behaves like any other DAS source when a reading query is submitted, therefore it is the client who decides when and what to do with this information. For the Dasty2 writeback extension, the user has three different modes to operate (Left side of Figure 5.4):

Disable the writeback display The first mode essentially ignores the writeback information and in this case Dasty2 just collects and displays the original information from the sources. This is useful for the users who do not want the collaborative information.

Writeback as an extra source Dasty2 can display the information coming from the writeback server as an extra data source. In such a case, all the writeback features will be displayed as new tracks, allowing the users to compare the original annotation with the last version of it in the writeback.

Merging the writeback with the sources In this mode, the writeback annotations overwrite the original ones in the graphic. This generates a similar graphic for features as was used in Dasty2, but with the modifications that the writeback server contains. The features tagged as deleted will be hidden in the graphic.

In the last two cases a list of the features tagged as deleted is displayed under the writeback panel in order to have an access point to edit those features.

An extra reading feature that has been added to Dasty2 is the option to recover the historical information for any of the writeback features. Figure 5.5(d) shows the tab that contains the list of all the versions that the selected feature has in the writeback server.

5.5.3 Writing in the Writeback

The writeback extension for Dasty2 allows the authenticated users to Create, Update and Delete features. The strategy is to reuse the internal pop-up windows of Dasty2 to display the information of a particular feature in order to provide the necessary tools to execute those functions in the same context as the selected feature. With this goal in mind, a set of tabs was added to those windows. Figure 5.5 shows the contents of the four different tabs that the user can choose after clicking on a particular feature. The first tab (a) is the detailed information that Dasty2 provided to the users for the chosen feature, the other three tabs give access to the writeback capabilities.

Below is a description of how the writeback capabilities are available in Dasty2:

Update Figure 5.5(b) is a screenshot of the edit tab; in it the user has the same detailed information, but in a form that allows the user to change the values of any field. When the information is sent to the server, it is stored as the current version of the feature and it will be the one to which the server returns for future requests.

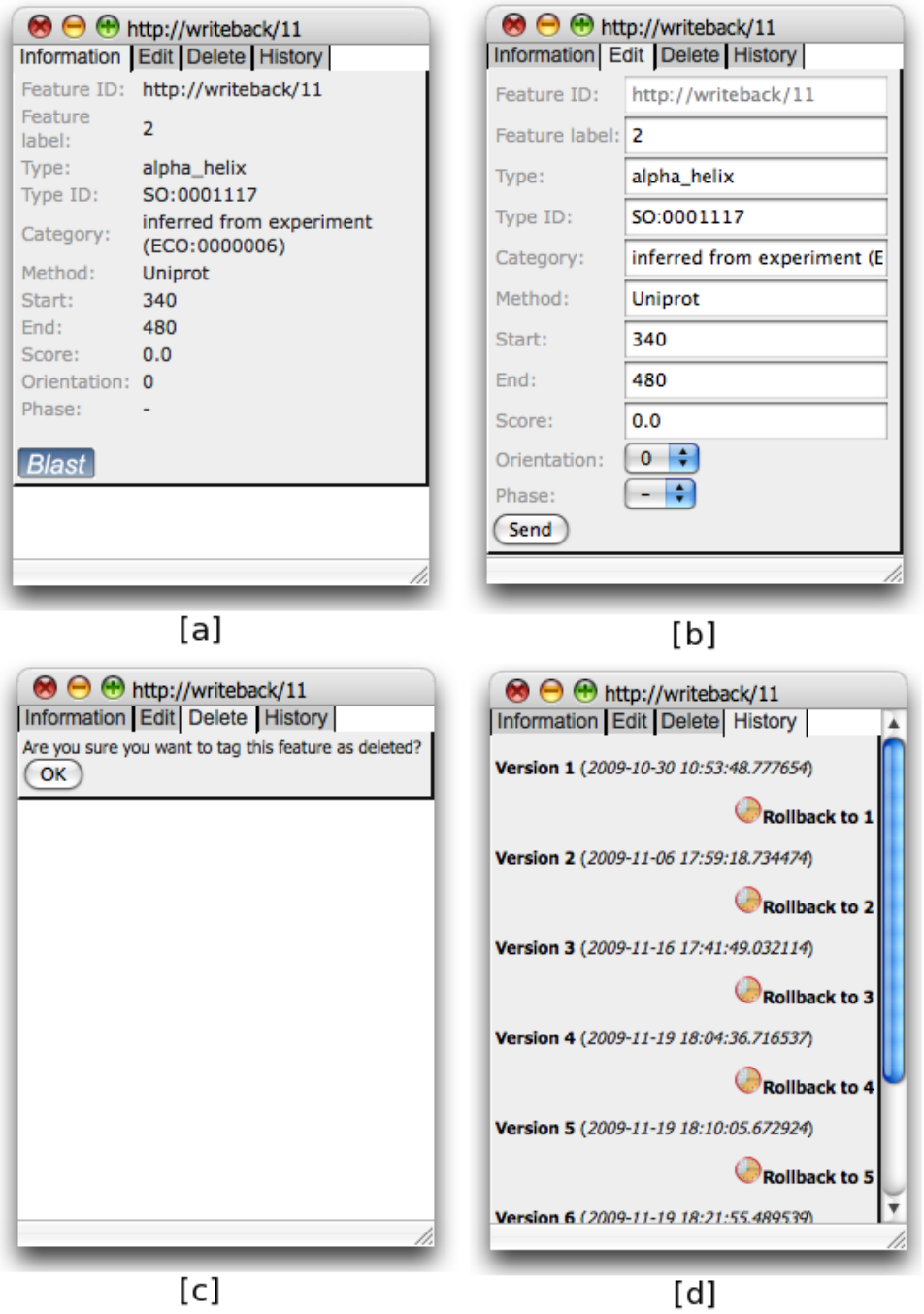

Figure 5.5: Tabs for writeback functions in Dasty2. (a) Detailed information of the feature. (b) Form to edit any detail of the feature. (c) Confirmation for deletion. (d) Writeback history of the feature

Another way to edit a feature is through the history tab (5.5(c)). In this case, the user can choose to roll back to a previous version. The "rollback" is not part of the writeback specification, what really happens is that the client requests an update with all the details of the previous version. This strategy was preferred over a complete rollback so that the previous versions do not get lost.

Create In the top-right corner of the writeback panel (Figure 5.4) there is a button to add a new feature. It opens a window similar to the one in Figure 5.5(b) but without any content in the fields. Therefore the user can create all the details of the feature in that window, send them to the writeback server and a new feature will be created.

Delete Figure 5.5(c) shows a confirmation message for the deletion of the feature. Features are not really deleted from the server, they are just tagged in such a way that this information can be used to hide the features in the *merge* method.

5.5.4 User Interface Aids

Version 1.53E of the DAS specification recommends the use of ontologies in order to standardize both annotation labels and evidence codes, and make the task of integrating annotation from several servers easier. The recommendation says that for the values of the attribute ID and the content of the element TYPE, the Biosapiens Ontology[4] ID and its corresponding name, should be respectivelty used. In the case of the evidence code, the ontology to use is the Evidence Code[5] [43] [23].

Looking to promote the use of those ontologies, a list of suggested terms from the corresponding ontology is displayed in the update form (Figure 5.5(c)) while the user is writing in the fields type and category; and then the user can choose from the list and the fields type, type ID and category are auto-completed in the right format.

The same form has a set of logic validations to ensure that the coordinates of the annotation are not out of the limit imposed for the size of the protein, and that the start amino acid is before the end amino acid. Finally, the orientation and phase are drop down lists to reduce the options to the permitted ones.

[4] http://www.ebi.ac.uk/ontology-lookup/browse.do?ontName=BS 2009
[5] http://www.ebi.ac.uk/ontology-lookup/browse.do?ontName=ECO 2009

5.5.5 User Stories

In order to elucidate the main features of the *Dasty2+writeback* application the following set of user stories have been defined. An explanation of how to do the user story in the application has been added.

1. *As an anonymous user I want to change the visualization mode for writeback information:* After all the sources (including the writeback) have been loaded the user can go to the Writeback panel (Figure 5.4) and choose between 3 different modes. The first option is to use the information of the writeback to modify the graphic and overwrite the original annotations for those in the writeback server. The second option is to consider the writeback as another source and display its features as in new tracks. The last one is to ignore the writeback information and only visualize the information coming from the original sources.

2. *As an anonymous user I want to create a user so that I can use the writeback writing function:* In the writeback panel (Figure 5.4) the user can click on the link "New User", A window requesting username, password and confirmation of the password is displayed. After submitting those fields, a user is created in the writeback server, and a session is started in the client.

3. *As an authenticated user I want to add a feature of the current protein:* In the Writeback panel (Figure 5.4) there is a button to create a new feature, after clicking this it will open a popup window similar to Figure 5.5(b) with all the fields empty. The user should fill in the information and after submitting it, the feature is created in the writeback server, and the graphic is repainted in order to include the new feature.

4. *As an authenticated user I want to edit a feature of the current protein:* When Dasty2 is displaying the graphic of positional features (Figure 2.5) the user can click on any of the features and a popup window will be displayed. This window contains the four tabs of Figure 5.5. Selecting Edit (b) a form where the information of the feature can be modify is shown. After submitting this information, the writeback server creates a new version of the annotation and the graphic is repainted to include the changes.

5. *As an authenticated user I want to delete a feature of the current protein:* Following the same procedure as the previous case the deletion tab can be displayed (Figure

5.5(c)). When the user confirms the deletion, a message is sent to the server and the feature is tagged as deleted, and can be visualized in the list of deleted features in the writeback panel (bottom of Figure 5.4).

6. *As an authenticated user I want to interact with the history of a feature of the current protein:* When a feature has been modified more than once, the user can see the different versions through the History Tab (Figure5.5(d)). Each version has a *Rollback* link, that can be used to put the information of that version as the current one for that feature.

5.6 Discussion and Conclusions

The developed software as a proof of concept has demonstrated that it is possible to add feedback functionalities to an existing DAS client, reusing its design paradigms and making the new options look like native operations of the software.

Dasty2 was extended not just to visualize information coming from a writeback server, but also to interact with this server creating and/or modifying annotations. The use of Ajax and Object Oriented JavaScript allowed us to develop a set of interfaces as close as possible to the one that can be created for a stand-alone application. Therefore, the same process can be applied to any other DAS protein client. In the case of other coordinate systems, extra considerations should be taken into account, especially in the case of hierarchical features, for example the Gene-Exon relationship, where for example, a chromosome can have annotated a gene X that is annotated to be composed of the exons A, B and C and another gene Y annotated to be composed of B and C.

The writeback client development process made some issues in the writeback server evident that were not detected during the design and implementation of its first version. For example, the strategy to authenticate a user was modified as described in 5.5.1 in order to simplify the process that the clients require to implement.

A feature that was added to the DAS server because of the development of the client was the possibility of removing the *DELETE* meta-annotations. It was necessary for the cases where the deletion was tagged directly from the original feature because in such cases there is no way to go back to previous states and the deletion is therefore permanent.

Although several improvements can be made to the *Dasty2+writeback* application in order to create policies of trustworthiness, like the ranking of features and users; we consider that the use of a third party server that required authentication plus the possibility to rollback to previous versions, are characteristics of the system that generate a good starting point for a community interaction.s

Chapter 6

Usability Experiment

6.1 Introduction

Our experiment is based on a working prototype of the DAS writeback client and we used a method of Formative Evaluation, looking to improve the usability of our System and verify that feedback operations can be excuted for users without specialized training.

A Constructive Interaction Experiment was designed, executed and analysed with the objective of improving the usability of the Collaborative Annotation System Interface. A description of the experiment, the method used and the results obtained are reported here.

This chapter is organized as follows: first a description of Constructive Interaction– the method chosen for the test, second the defined conditions and rules for the experiment, third the results obtained from the experiments; and finally a discussion with the conclusions of the process.

6.2 Choosing an Experiment

Our working prototype consists of two components: a server and client, both of which are extensions of widely used pieces of software. The server extends the MyDas application,

which is used to provide DAS features for important projects, such as UniProt[1]. For the client, we extended Dasty2, an application for which the main installation is at the EBI-EMBL site[2], and which have also been included as an extra component in other projects such as UniProt[3]. Considering , their wide use, the robustness of the extended applications (MyDas and Dasty2), and in the base performance has already been proven, so we do not need to test these here.

The server stores all the information using the PostgreSQL RDBMS, delegating all the persistent storage low-level tasks to that Management System. The reason for this, is that we believe with an appropriate installation of the server it will be able to handle the required information load and resulting traffic. This means we can focus exclusively on the testing of the new capabilities related with the writeback, especially since a writeback server is not expected to hold as much information as a centralized source.

As described in 6.3.2, the available sample of users for the experiment does not have any previous experience in usability tests. This inspired us to consider alternative methods to the Heuristic Evaluation.

Think-Aloud protocols provide the opportunity to view a working prototype in action, capturing some of the potential issues that an interface experiences, using different methods, such as, videotaping the sections for its posterior analysis, related to what the user wished or expected the system to do in comparison to the real actions executed by the system and the respective responses.

However, some of the criticisms of traditional Think-Aloud protocols suggest an unnatural behaviour of having to consciously express their actions, their motivation and the expected outcome. An alternative which tries to address this criticism is Constructive Interaction. In this cases two users utilising a single machine, would have to talk to each other in order to define the strategies necessary to solve the tasks.

The obvious advantage to using the constructive Interaction method would be the understanding of the collaboration between partners, which is very relevant in cooperative environments [28]. This project proposes the case of the Collaborative Annotation System and that is the main reason for choosing a Constructive Interaction method as the software

[1] http://www.ebi.ac.uk/uniprot-das/ 2009
[2] http://www.ebi.ac.uk/dasty/ 2010
[3] http://services.uniprot.org/uniprot-dasty/client/uniprot.php 2009

evaluation technique.

6.3 Experimental Design

6.3.1 Test Object

The Collaborative Annotation System proposed in this project is the focus of this test, although it is obvious that the client of the system receives more attention in the test given that it interacts directly with the user. The server is also an important part of the test because the interfaces in the client are the contact point between the user and strategies defined for the system and implemented in a third party server.

The client is a web application called *Dasty2+Writeback*, which makes extensive use of Web 2.0 technologies. The client is an extension of an existing application, therefore the experiment will test the new features as well as validate the previous characteristics of the system which have not been modified.

6.3.2 Subjects

The group of users selected for the test consisted of eight postgraduate students from the Computational Biology group at the University of Cape Town. This ensured that the users had both good skills in the use of web applications and biological knowledge to make use of the application; however none of them have participated in a usability experiment before, and therefore their expertise in usability testing is limited. The age of the youngest test user was 27 years old and the oldest is 32 years old with an average of 29.6 and a standard deviation of 2.0. Five of them were MSc and three were PhD students of bioinformatics. Seven males and one female took part in the experiment. Although they have been exposed through presentations to what the system is, and the technology behind it, none of them had used the system before. They knew each other and the dyads were selected randomly. Even though we recognize that 4 experiments is a small number of tests, the group is very representative of the final users of the system and we are therefore confident that the results of the tests should be conclusive.

6.3.3 Tasks

In order to define the tasks that the user has to accomplish during the experiment, a real publication that reports positional annotations was used [31]. The reported annotations are out of date in comparison with the ones on the DAS servers; however it makes this information appropriate for the test because it ensures that the information recovered by the client will be different to the one in the paper and therefore modifications will be necessary.

The use of a published paper that contains features of an existing protein generates the necessary confidence about how representative the test is of the scientific environment where the application is expected to be used. Other test environments could have been chosen, such as, a *"jamboree"* project, where a particular biological sequence is the focus of all the annotators. However, in this scenario the creation of features would be the most often used writeback capability (if not the only one), and then the experiment could be unbalanced in just a few of the features being tested– This is the reason why we decide to use the published paper.

From the publication, a group of annotations were selected for the test and specific tasks that involve those features were defined in order to enforce the use of the different writeback capabilities (Add, update, create and delete). The test tasks looked to be representative of the use of the *Dasty2+writeback* capabilities. This is not a test that evaluates all the functionalities, however it tries to be representative of the most important ones.

A help page that contained the explanations of the different modules of the *Dasty2 +writeback* application was provided to the users in order to ensure the availability of information that allows them to solve the proposed tasks.

The first two tasks were designed to familiarize the user with the information contained in Dasty2 as well as how to manipulate those results to filter and visualize only the desired information. The third task involved a requirement of authentication for the use of the writeback, which correspond to the *User Story 2* (Section 5.5.5). Tasks 4 and 5 were created to use the writeback features, The forth one looks for the user to go throug the User Story 3 and the fith task is related with emphUser Story 4 (Section 5.5.5).

Below is the set of tasks that the users had to perform during the experiment:

1. In the web browser go to the Dasty2 URL[4] and query the information in all the available sources for the protein with the accession number O14737.

   When all the servers have provided the information, please fill in the following fields. (All this information is contained in the response page):

   - Sequence length:
   - First 10 amino acids:
   - A server that has provided annotations:
   - A server that does not have feature annotations available for this protein:
   - A server with errors or warnings (if any):
   - Any name of the protein:
   - A publication citation about this protein:

2. Manipulation and filtering options:

   - Make sure that the positional features graphic is displaying the columns type, server and category.
   - Filter the graphic in order to display all the *poly peptide secondary structure* related features, coming from any server but *uniprot_aristoteles*, with evidence code *inferred by curator* or *inferred from in-silico analysis*.

3. Create a writeback user log in.

4. The following extract of the paper [31] contains information about the structure of the protein O14737 that is not in your current graphic. Add those features to the graphic.

   "In this study, the heteronuclear NMR method was adopted for understanding the solution conformation of human PDCD5 protein. The 3D solution structure of PDCD5 protein is supposed to be divided into three structural regions, a rigid core region and two dissociated terminal regions. The core region (N41–Q101) consists mainly of a triple-helix bundle. The N-terminal region (D3–R40) is an ordered, but not a rigid, structural region which contains abundant secondary structures, and packs very

[4] http://oware.cbio.uct.ac.za/~gustavo/client

loosely against the core. The C-terminal extension represents a mobile and unstructured region (Q102–D118) with a dynamically frayed tail (S119–Y125) in the protein that may be capable of interactions with nucleic acid."

5. The alpha helices reported in [31] are summarized in the following table. Create, Update and/or Delete features in order to adjust the DAS information in the graphic with the one in the table.

Table 6.1: Reported alpha helices

Alpha helix	Start	End
α 1	3	19
α 2	26	33
α 3	41	46
α 4	50	61
α 5	63	79
α 6	89	100

Note 1: It is preferable to update a feature than delete one and then create a new one.

Note 2: After finishing this point, delete any extra helices in the graphic that contradict or replicate your annotations.

6.3.4 Questionnaire

After the usability sessions, the users completed a form about demographic details such as age, gender and education. It also contained questions about the experiment itself, the method used, etc. Finally, it provided an extra chance to add any comment or suggestion about the system. The full questionnaire can be found in appendix A.

6.3.5 Experimental Procedure

Every user was introduced to the experiment and had the procedure explained to them, with the following facts emphasized:

- This is an evaluation of the software and not of the user.

- Errors, problems or difficulties are expected in the application, so, please indicate any of those.
- The test results will affect the final version of the application.
- Please avoid talking about the system outside of the experiment to prevent any biased behaviour of potential participants of the experiment.
- The test is voluntary and can be stopped at any time for the participant.
- Any record of the test is confidential and will be used only for the purposes of the experiment.

The experimental procedure was the following: The participants were seated randomly at the computer, one of them sitting in front of it (Actor), the other next to it (Co-actor). Subsequently they received the instruction :

- Working together, which in this case means that any executed action should first be consulted between them.
- Try to make the ideas to solve the problem explicit.
- Please do not interact with the facilitator.

On finishing the task, they were required to fill in the questionnaire.

6.3.6 Processing the data

All the sessions were recorded in two different ways: with a video camera and by using *Screen Record 2*[5], a software tool that records all the actions that happen on the screen as a Quicktime movie.

The video camera enabled us to capture any expression or body gesture that indicates the approval or disapproval of components of the software at the moment of executing the tasks. The screen video allowed us to have a detailed record of all the mouse movements

[5] http://www.miensoftware.com/screenrecord.html 2009

and any introduction of text in certain areas of the screen, which in our case was the browser window. Both methods recorded the audio of the session.

After all the sessions were executed, the records were studied by synchronizing both videos and analysing them to detect when the user was having problems in solving any of the tasks or when they expected the functionalities to work in a different way.

The first part of the questionnaire contains demographic information and their opinion about the test itself. The demographic information was discussed in Section 6.3.2 and is summarized in Table 6.2.

Table 6.2: Demographic Information of the experiment users

User #	Age	Gender	Completed Degree	Current degree	Role
1	27	M	BSc	MSc	Actor
2	28	M	MSc		Co-actor
3	27	F	MSc		Co-actor
4	30	M	MSc	PhD	Actor
5	31	M	MSc	PhD	Actor
6	31	M	Postgraduate	MSc	Co-actor
7	32	M	BSc	MSc	Actor
8	31	M	Postgraduate	PhD	Co-actor

Table 6.3 condenses the information about the experiment perception of the test subjects.It is interesting that 75% of the users considered that the level of difficulty of the experiment was Medium and 25% even considered the test easy, however only two of them affirmed to answer all the questions correctly. Constructive Interaction, as the chosen technique for the experiment, did generate a good impression in the participants. All of them considered that the team work helped to solve the tasks, mainly because of the complement of knowledge that the partners can bring, especially biological knowledge. Their opinion about how the speed of the work is affected by working in a team was divided: 50% considered it faster, 25% about the same and the other 25% considered it to be slower to work as dyads. Despite this, the comments about this topic reinforced the good perception of the experiment, for example one of the users said *"Slower but probably more correct"* and another one wrote *"two people trying to use one computer = slower... but more ideas = faster"*.

Table 6.3: User considerations about the experiment

User #	Difficulty	Correct	Team work help	Team work speed
1	Medium	Most	Yes	Faster
2	Medium	No	Yes	About the same
3	Medium	No	Yes	Slower
4	Medium	No	Yes	Faster
5	Medium	Most	Yes	About the same
6	Medium		Yes	Faster
7	Easy	Yes	Yes	Slower
8	Easy	Yes	Yes	Faster

6.4 Results

A summary of the analysis of the session by task can be found in the appendix B. From there, a set of usability issues were compiled. As Rolf Molich suggested in [35], the comments and issues of the usability test were classified as disasters, serious problems, minor problems, positive findings, bugs, and suggestions.

An issue was considered a *Disaster* if it breaks the execution of the program or causes an abrupt interruption of the server. A *Serious Problem* is when one of the main features of the application generates completely different results to those planned. When the application generates the correct results, but the procedure to get the result is different to what the user tries, or the messages confuse the user, the issue is catalogued as a *Minor Problem*. A *Positive Finding* is annotated when all the groups have used a new feature without any inconvenience, or when a user explicitly shows his appreciation for a feature. If a functionality works well most of the time, but there is a sequence of events that evidence an error, it is considered to be a *Bug*. Finally, a *Suggestion* can be made by a verbal or a written comment.

As explained previously, the first two tasks of the experiment were about pre-existing functionalities of Dasty2, and they aimed to familiarize the user with the interface and detect any usability issue with the current interface. The other three tasks were related to the writeback functionalities; the actual focus of this project. Due to this, the set of errors, usability issues and/or suggestion were divided into Dasty2 and *Dasty2+writeback* related issues. The terminology about the Dasty2 Panels and components is explained in Section 2.3.4.

6.4.1 Dasty2 issues

Disasters : None

Serious Problems : None

Minor Problems :

1. The *Registry label* (Figure 6.1(A)) was not taken into consideration at the time of submiting a query (*3 Group*).
2. Users do not know what kind of information can be found in the *Non-positional feature* panel (*1 Group*).
3. The *filtering trees* (Figure 6.1(B)) are so big that the user can not see the graphic, and therefore the updating "on the fly" behaviour is not noticed (*3 Groups*).
4. The user does not realize that the *Annotation server* link (Figure 6.1(C)) at the top of the progress bar is a link to show the log of the loaded servers (*2 Groups*).

Positive Findings :

1. The manipulation options are easy to find and use (*4 Groups*).
2. Sorting the list of non positional features was useful to solve task number 1 (*1 Group*).

Bugs :

1. The *filtering trees* (Figure 6.1 (B)) do not work properly when more than one is affecting the graphic. The filters are working in an isolated way overwriting a preselected filter instead of adding a restriction to the filter (*3 Groups*).
2. Some features do not show the popup window when the mouse is over it. (*1 Groups*).

Suggestions :

1. Messages informing users that the graphic has been updated after changing the filtering options.

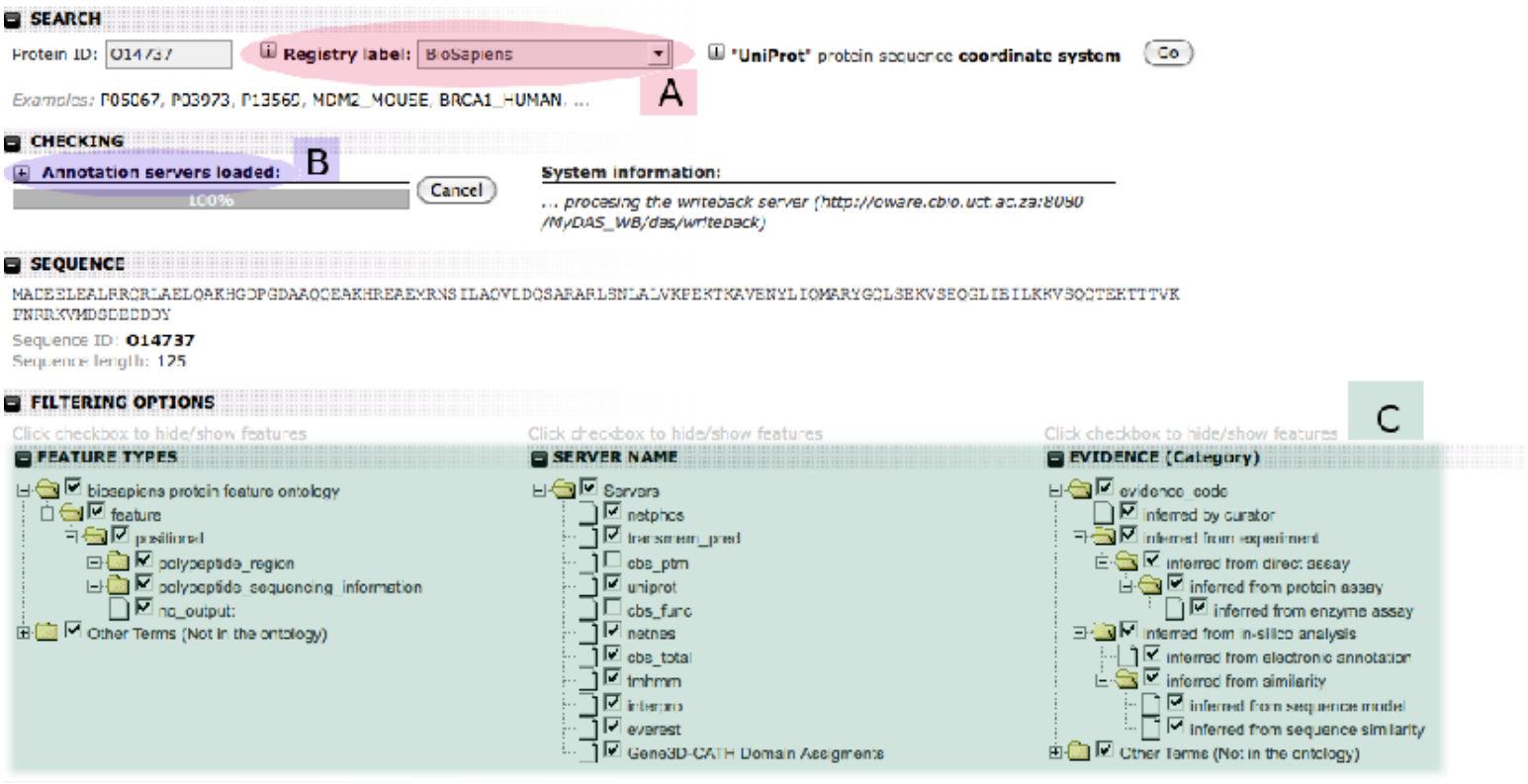

Figure 6.1: *Dasty2 Minor Problems*: Areas with a minor problem detected. (A) Registry Label Chooser. In the experiment it was expected that the users choose the option "Any" to ensure all the servers were queried. (B) Annotation Server Link. When Clicket a log of the status of the requested servers is listed. (C) Filtering Trees. As a result of the experiment it was detected that the filtering option does not work properly when using more than one tree.

2. The volatile popup window that is displayed when the mouse is over a feature should disappear when the permanent popup is requested.
3. Manipulation options should be closer to the graphic, it could even be put inside the same panel.
4. Make the filtering trees (Figure 6.1 (B)) as popup windows to allow the user to see the graphic while the tree is being manipulated.

6.4.2 Dasty2+Writeback issues

Disasters : None

Serious Problems :

1. The graphic does not draw the first annotation created for a protein. The query had to be submitted again and it worked well from then onwards (*4 Groups*).

Minor Problems :

1. The popup window to create a user (Figure 6.2(A)) should disappear automatically after a user is created. When it is still open, it creates the sensation that the user is not created (*4 Groups*).
2. The meaning of the feature fields (Figure 5.5(b)) is not clear (*4 Groups*).
3. The first try to get the writeback options of a feature was using right click instead the left click, that is the way it currently works (*4 Groups*).
4. The message to inform you that a user was created is in the form of a warning, generating the impression that an error has ocurred (*4 Groups*).
5. Type ID (Figure 5.5(b)) causes problems to the users when the type is out of the ontology because they do not have an ontology Id to fillin the field, but the field is still mandatory (*3 Groups*).

Positive Findings :

1. All the groups figured out how to add, create and delete features.
2. The distinction between the original feature and the new version in the *Render as new tracks* visualization mode was intuitive for all the users.

Bugs :

1. The automatic login after user creation was not working. The login panel behaved as if the user was logged in, but all the writeback functionalities were still locked (*4 Groups*).
2. In the history of a feature, the current one should not be displayed (Figure 5.5(d)).

Suggestions :

1. Add tool tips for the fields in the forms to create and edit a feature (Figure 5.5(b).
2. Add a function to copy or duplicate an existing annotation in order to re-use most of the values of the fields and just edit the ones that change.
3. Change the name of the link to remove a deletion to *Restore*
4. Allow the use of the *Enter* button to submit the data for a created/edited feature.
5. Allow the use of the keyboard to navigate the component to display suggestions based on the ontology terms.

6.4.3 Corrective Measures

The correction of issues strictly related to Dasty2 was out of the scope of this project, nonetheless, thanks to the close collaboration between the current Dasty2 development group and ourselves, a report of issues and comments was submitted to them and the solution of the issues and implementation of the suggestions will be considered as part of the 2010 Dasty2 maintenance plan.

The writeback related issues were solved in the same order that they appear in the list to give a higher priority to the serious problems, then the minor problems, bugs and finally the suggestions. The items related to a keyboard friendly navigation were cataloged as low priority because they are a desirable feature but the user still has the option to use the mouse for those actions. Below is the report of the changes during this development cycle, following the same numeration of that in section 6.4.2.

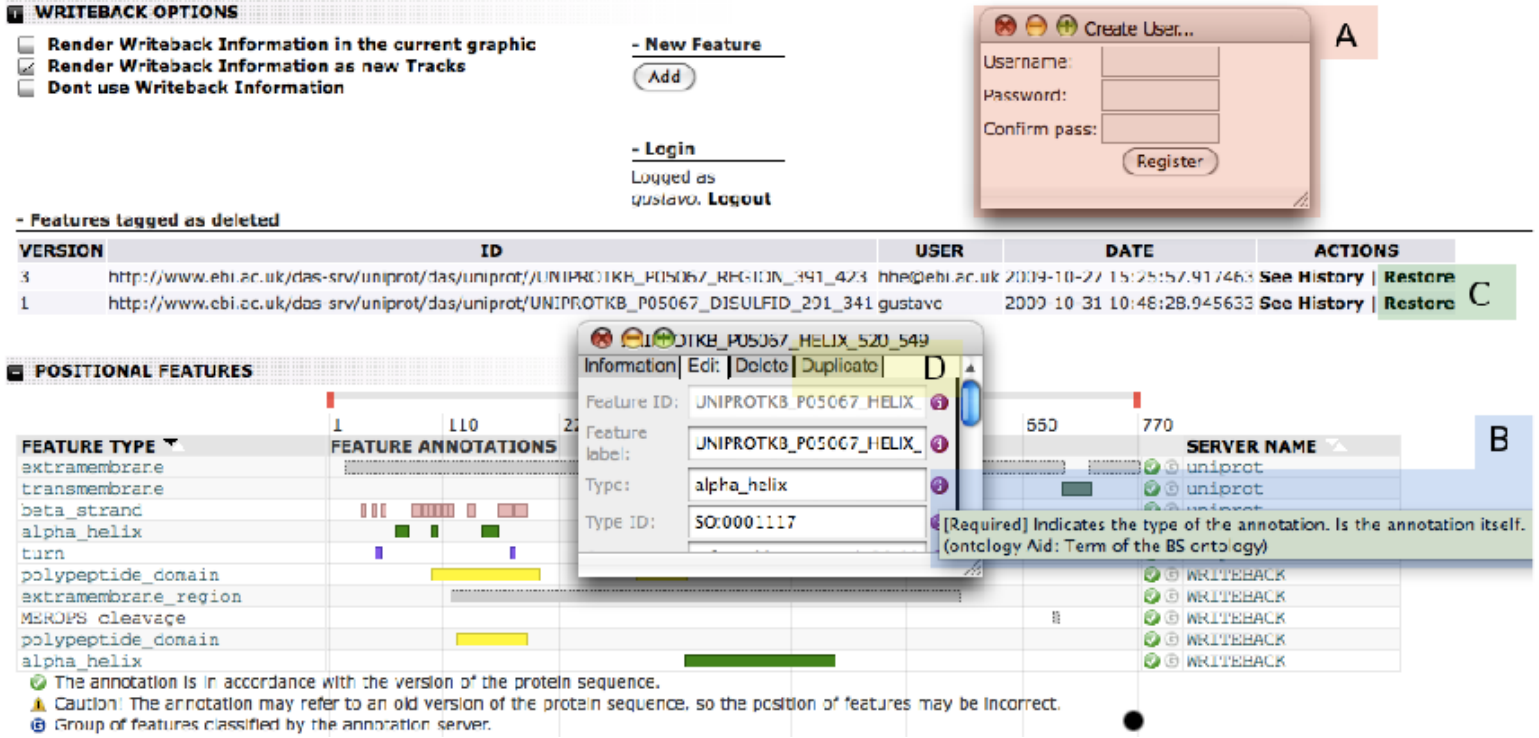

Figure 6.2: *Dasty2+writeback Corrections*: Areas with a corrected issues. (A) Create User Form. (B) Tool tip and icon help for fields in the Features Edit Tab. (C) Restore Link for deleted features. (D) Duplicate Tab to copy all the fields of a feature to a new one.

Serious Problems :

1. When a protein does not have features in the writeback server, this will return an *Unknown segment* in the document. The problem was that at the time of creating or editing a feature for the first time in a protein, the client tried to add that feature to the *Unknown segment*. The correction was done and now the document is replaced completely, not just the feature, to include a recently created segment.

Minor Problems :

1. Now the "Creat User" popup window (Figure 6.2(A)) closes automatically after the user is created.
2. An icon for information (Figure 6.2(B)) is now displayed at the side of the fields, and if the mouse is over it, a help message with details about the field will be displayed.
3. Now both buttons are displaying the writeback tabs.
4. Dasty2 displays the messages in the System information panel, so now the "user created" message is displayed there.
5. Some servers that were providing information before the implementation of the ontologies in DAS solved this issue by using the same Type term as Type Id, so we decided to follow the same strategy for the types out of the ontology.

Bugs :

1. The cause of this bug was that given that the *Create User* popup window (Figure 6.2(A)) did not disappear after creating the user (Minor Problem #1). The participants of the experiment pressed the *submit* button again and the second request was a failure because the user already existed, causing the logout. By hiding that window at the time of the creation, this bug was solved.
2. Now the *History* panel only displays previous versions.

Suggestions :

1. Same as Minor Problem #2 (Figure 6.2(B)).

2. A new tab was added to the writeback options called Duplicate. It looks the same as Figure 5.5 but instead of triggering the *Update* of a feature, it sends a *Create* request, and then the desired functionality is reached.
3. The link to undo a deleted feature was renamed to Restore (Figure 6.2(C)) .
4. Postponed for a second maintenance cycle.
5. Postponed for a second maintenance cycle.

6.5 Discussion

The major outcome of the experiment is that the users were able to use the writeback functionalities without extensive training. This raises two important points worth highlighting: Firstly, both server and client are doing what the user expects of them, and secondly, the functionalities are intuitive enough to allow untrained users to solve protein annotation tasks. We believe these features are the main requirements for a system which provides support to a collaborative annotation community.

The duration of the sessions varied from forty five minutes for the second group to one and a half hours for the first group. The observations yielded from the sessions were such that the group containing subjects from different backgrounds had a better performance. For example, one subject possessing a greater biological knowledge base and the other subject having a background rooted in computer science. Having a basic knowledge of DAS also proved to be useful, for instance, one of the groups knew that information such as the name or publications related to the protein are presented as *Non positional features* in DAS, useful information for the first task.

A constant obstacle the groups faced, was they did not understand the meaning of the fields to create/update a feature, the strategy for most of them was to have a look at existing annotations to check what kind of information corresponds to each field– even then, some of them are clearly different to the expectations of the user, for example one user comment was *"Method is UniProt?? That's weird"*.

An interesting discussion was held by the users in group 4. One of them was interested in testing the software provoking errors, for instance placing characters in numeric fields;

whilst the other user focussed on addressing the tasks in the best way possible. We consider both positions to be valuable in testing a procedure. It should also be noted that the facilitator did not interfere with the users performance. They finally behaved in a hybrid way, looking to solve all the tasks, but sporadically, they committed errors on purpose to test the robustness of the software.

As expected, an analysis of the sessions indicated several usability issues that were undetected during the design and implementation process. On the one hand, solving the detected problems ensures a functional and usable system. On the other hand, ideas and suggestions can improve the speed of manual annotations, and help inexperienced users.

For these reasons we consider the experiment to be a success as it reinforces our hypothesis which stated that the developed system will provide an adequate environment to cooperate during the annotation process.

Chapter 7

Concluding Remarks

The objective of this research was to create a method to annotate proteins in a collaborative environment where the consumers of the information have the option to become authors of new annotations or edit the existing ones. Such an ecosystem can contribute to part of the curation of automatic annotation as a community process and simultaneously as a quick way to publish manual annotations while these are in the queue of a curated database awaiting annotation. This is currently not possible.

The Distributed Annotation System, DAS, was chosen as the starting point to reach this goal. It provides an existing distributed platform with more than 600 registered sources by the end of 2009 (110 for protein annotations), and consequently a large number of users that are potential members of the annotation community; moreover, the active development community of DAS were always willing to aid the project by providing valuable advices and suggestions.

DAS writeback is the name of our proposed system, which is capable of handling reading, writing, editing and deleting requests from the users of DAS. In order to design and develop such a system it was necessary to first design the supporting architecture for the new features, and then, to define an extension of the DAS specification, and implement server and client components. All of these milestones were completed while trying to maintain the existing style contained in DAS technology, looking for an easy adoption of the system.

This summarizes our methodology which is best described as: Experimental Computer

Science. This method requires the building of computational artefacts which are then evaluated empirically.

The designed architecture proposed the addition of an extra server in DAS with writeback capabilities for each coordinate system. Confidence in the information provided for an annotation system is a crucial objective. DAS deals with this by allowing the users choose the source of the information. Ultimately it is the user who decides which group of DAS servers to query, by choosing DAS Registry Labels. This strategy, allows the users to choose what was adopted in this project. Therefore the information emerging from the collaborative annotation server is optional, it never overwrites an existing annotation, it is an extra layer that can visually change the annotation graphic but the original source is never modified. For this reason we proposed an independent server to manage the community annotations.

We proposed an extension of the DAS protocol inspired by the RESTful architecture principle called *Uniform Interface*; the created specification indicates use of the HTTP methods GET, POST, PUT and DELETE as the interface for the main commands of a writeback DAS server for reading, creating, editing and deleting, respectively. The format of the message of all the commands is the DASGFF, which is the standard language for DAS in order to retrieve annotations.

The solution is the product of a User Centred Design process. The defined architecture and protocol extension were extensively discussed through the worldwide specialized online forums of DAS[1][2][3], getting important feedback from experts. In addition, a presentation about a writeback prototype was made at the DAS workshop in 2009 [45], creating enough interest to open a discussion group on the last day of the workshop[4].

The knowledge gained there was particularly relevant and focused and ultimately allowed for the modification of our architecture and provoked the creation of an entirely new protocol extension. Our specification was published on the official website of DAS [5] to be considered as an addition in future releases of the DAS protocol. Our resulting solution is original, it is also compatible with the current DAS protocol, and moreover, it is extensible

[1] http://lists.open-bio.org/pipermail/das/2008-October/thread.html 2008
[2] http://lists.open-bio.org/pipermail/das/2008-November/thread.html 2008
[3] http://lists.open-bio.org/pipermail/das/2009-June/thread.html 2009
[4] http://www.biodas.org/wiki/DASworkshop200903Day3 2009
[5] http://www.biodas.org/wiki/DAS1.6E#DAS_writeback 2009

to future versions.

An extension of a DAS server was implemented in order to support our writeback extension of the protocol. Several DAS servers were studied and MyDas was chosen to be extended. A writeback data source was implemented that stores the annotation in a database that has annotations as its main entity, and any edition or deletion of a feature becomes a new version of it.

Dasty2 was extended with the functionalities to interact with the writeback server under the rules of our proposed specification. The *Dasty2+writeback* client visualizes the community annotations as separate tracks or replaces the original annotation. The user can also choose just to ignore any writeback information. The writeback capabilities (new, edit or delete) can be accessed via the current Graphical User Interface, allowing the user to create information in the context of the existing annotations.

At the conclusion of two cycles of design, implementation and feedback from the global community, we subjected the system to a final formative evaluation. The experimental system was installed on an Internet server at the University of Cape Town. This installation was used to execute a usability experiment, which demonstrated its potential for real biological applications, because all the tasks of the experiment were extracted from a published paper. The technique used to design such an experiment was Constructive Interaction and it was executed with the participation of eight postgraduate students organized in dyads. All the sessions were recorded and analyzed.

The experiment revealed fifteen usability issues, of which only one was a *Major Problem*. The remaining five constituted *Minor Problems*. The experiment revealed two *Positive Findings*, two *Bugs* and five *Suggestions*. All problems and bugs were consequently solved prior to the completion of the final version of the application. Three suggestions were implemented; however two suggestions have been postponed with the intention of integrating them during a maintenance cycle in the future.

The experimental system is now available in a live web server [6] and the source code is freely available through SVN for both server[7] and client[8].

[6] http://oware.cbio.uct.ac.za/~gustavo/client/biosapiens.html
[7] https://mydaswb.svn.sourceforge.net/svnroot/mydaswb/MyDAS_WB
[8] https://dasty.svn.sourceforge.net/svnroot/dasty/branches/dasty_wb

The experiment vindicated our User Centered Approach. The one major issue has been corrected. In general we demonstrated the usefulness of our concept. All the groups that participated in the experiment were able to Create/Update DAS annotations from a published paper, we consider this fact to demonstrate that our system is effective, usable and will provide the appropriate environment for the creation and evolution of a protein annotation community.

The system also possesses the potential to be installed in Intranet environments which have a restricted community of users; cooperatively working on draft versions of the annotations. It is possible to only release those annotations to the general community when the team considers they have reached the desired quality level.

Server and client implementations were created following version 1.53 of the DAS protocol– an update of those components is suggested as future work in order to be compatible with DAS in its recently released version 1.60.

Other DAS servers and clients can be extended following the writeback specification, the implementations presented in this project can be seen as proofs of concept of the whole system and several improvements can be made in order to have a more usable and effective tool. We propose future developments could include the implementation of filters by dynamic trust rankings based on both features and users, this will achieve a higher level of confidence in the information of the writeback system.

We consider that our approach for this project was correct, highlighting the involvement of the DAS community from the very beginning of the project, which kept us on the right track to reach our objectives. Another key point during the project was to reuse existing technologies (DAS, Dasty, MyDas, etc.) that have been proven to be robust and useful. It was also remarkable that all the feedback obtained from the usability experiment put the whole project into context, allowing us to detect the strong and weak points of the system.

An important milestone in the future is to provide the same technology for other types of genetic material, for example, a writeback for DNA information or for experimental information like microarrays.

Finally, the success or failure of any collaborative system is recognized through the interaction of real users with the system, and to be able to observe that, additional time is

required. We hope this system contributes to creation of a more public, easily updatable, and reliable protein knowledge base.

Bibliography

[1] B. Thomas Adler and Luca de Alfaro. A content-driven reputation system for the wikipedia. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 261–270, New York, NY, USA, 2007. ACM. 9

[2] Maristella Agosti, Giorgetta Bonfiglio-Dosio, and Nicola Ferro. A historical and contemporary study on annotations to derive key features for systems design. *Int. J. on Digital Libraries*, 8(1):1–19, 2007. 8

[3] Maristella Agosti and Nicola Ferro. A formal model of annotations of digital content. *ACM Trans. Inf. Syst.*, 26(1):3, 2007. 9, 28

[4] Maristella Agosti, Nicola Ferro, Emanuele Panizzi, and Rosa Trinchese. Annotation as a support to user interaction for content enhancement in digital libraries. In *AVI '06: Proceedings of the working conference on Advanced visual interfaces*, pages 151–154, New York, NY, USA, 2006. ACM. 9, 10

[5] Benedikte S. Als, Janne J. Jensen, and Mikael B. Skov. Comparison of think-aloud and constructive interaction in usability testing with children. In *IDC '05: Proceedings of the 2005 conference on Interaction design and children*, pages 9–16, New York, NY, USA, 2005. ACM. 28

[6] Marc Andreessen and Eric Bina. NCSA Mosaic: A global hypermedia system. *Internet Research*, 4(1):7–17, 1994. 8

[7] Umesh Bhatia, Keith Robison, Walter Gilbert;, Hans-Peter Klenk, Owen White, and J. Craig Venter. Dealing with Database Explosion: A Cautionary Note. *Science*, 276(5319):1724–1725, 1997. 2

[8] Greg Brown. Object oriented javascript, Jun 2006. 56

[9] Soren Brunak, Antoine Danchin, Masahira Hattori, Haruki Nakamura, Kazuo Shinozaki, Tara Matise, and Daphne Preuss. Nucleotide sequence database policies. *Science*, 298 (5597)(1333), November 2002. 11

[10] Maaike J. Van den Haak, Menno D. T. de Jong, and Peter Jan Schellens. Employing think-aloud protocols and constructive interaction to test the usability of online library catalogues: a methodological comparison. *Interacting with Computers*, 16(6):1153–1170, 2004. 27, 28

[11] Tobias Doerks, Amos Bairoch, and Peer Bork. Protein annotation: detective work for function prediction. *Trends in Genetics*, 14(6):248–250, 1998. 1

[12] Robin Dowell, Rodney Jokerst, Allen Day, Sean Eddy, and Lincoln Stein. The distributed annotation system. *BMC Bioinformatics*, 2(1):7, 2001. 3, 11, 13, 18

[13] 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. 17

[14] 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. 21

[15] Jesse James Garrett. Ajax: A new approach to web applications, 2005. 19

[16] Rich Gazan. Social annotations in digital library collections. *D-Lib Magazine*, 14(11/12), November/December 2008. 10, 29

[17] Google. Google data api protocol. Technical report, Google, 2009. 26

[18] J.C. Gregorio and B. de Hora. The atom publishing protocol. Technical report, NewBay Software, 2007. 26

[19] Asia Grzibovska and Andreas Prlic. DAS2 writeback server implementation. Master's thesis, Chalmers University of Technology, 2008. 25

[20] Wellcome Trust Sanger Institute. Dazzle, 2009. 17

[21] Wellcome Trust Sanger Institute. The pfam database, 2009. 21

[22] ISO. 9241: Ergonomic requirements for office work with visual display terminals part 11: Guidance on usability. Technical report, ISO, Geneva, Switzerland, 1998. 27

[23] 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. 13, 19, 51, 63

[24] 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. 25

[25] Ralph E. Johnson. Components, frameworks, patterns. In *SSR*, pages 10–17, 1997. 18

[26] Philip Jones and Antony F. Quinn. Mydas, 2008. 18

[27] José Kahan and Marja-Ritta Koivunen. Annotea: an open rdf infrastructure for shared web annotations. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 623–632, New York, NY, USA, 2001. ACM. 8, 29

[28] Helge Kahler, Finn Kensing, and Michael Muller. Methods & tools: constructive interaction and collaborative work: introducing a method for testing collaborative systems. *interactions*, 7(3):27–34, 2000. 68

[29] Cold Spring Harbor Laboratory. The lightweight distributed annotation server (ldas), 2001. 16

[30] Jason Levitt. Fixing ajax: Xmlhttprequest considered harmful, November 2005. 55

[31] Dongsheng Liu, Yingang Feng, Yuan Cheng, and Jinfeng Wang. Human programmed cell death 5 protein has a helical-core and two dissociated structural regions. *Biochemical and Biophysical Research Communications*, 318(2):391–396, 2004. 70, 71, 72

[32] Catherine C. Marshall. Annotation: from paper books to the digital library. In *DL '97: Proceedings of the second ACM international conference on Digital libraries*, pages 131–140, New York, NY, USA, 1997. ACM. 10, 29

[33] 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. 22

[34] Naomi Miyake. Constructive interaction and the iterative process of understanding. *Cognitive Science*, 10(2):151–177, 1986. 28

[35] Rolf Molich and Christine Perfetti. Usability testing best practices: An interview with rolf molich, July 2003. 75

[36] Barend Mons, Michael Ashburner, Christine Chichester, Erik van Mulligen, Marc Weeber, Johan den Dunnen, Gert-Jan van Ommen, Mark Musen, Matthew Cockerill, Henning Hermjakob, Albert Mons, Abel Packer, Roberto Pacheco, Suzanna Lewis, Alfred Berkeley, William Melton, Nickolas Barris, Jimmy Wales, Gerard Meijssen, Erik Moeller, Peter Roes, Katy Borner, and Amos Bairoch. Calling on a million minds for community annotation in wikiproteins. *Genome Biology*, 9(5):R89, 2008. 11, 29

[37] Jakob Nielsen. *Usability Engineering (Interactive Technologies)*. Morgan Kaufmann, 1st edition, September 1993. 27

[38] Jakob Nielsen and Rolf Molich. Heuristic evaluation of user interfaces. In *CHI '90: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 249–256, New York, NY, USA, 1990. ACM. 27

[39] Mary Riley Claire O'Malley and Stephen Draper. Constructive interaction: A method for studying user-computer-user interaction. In *Proceeding of IFIP Interact*, pages 269–274. ACM Press„ 1984. 28

[40] 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. 26, 48

[41] 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. 14

[42] Andreas Prlic, Thomas A. Down, and Tim J. P. Hubbard. Adding Some SPICE to DAS. *Bioinformatics*, 21(suppl_2):ii40–41, 2005. 21

[43] DAS Registry. DAS Ontology Extension, 2008. 63

[44] DAS Registry. Help on coordinate systems, 2009. 16

[45] Gustavo A. Salazar. DAS Writeback. DAS Workshop 2009, March 2009. 85

[46] Lincoln Stein. Genome annotation: from sequence to biology. *Nat Rev Genet*, 2(7):493–503, 2001. 10.1038/35080529. 2, 11

[47] Walter F. Tichy. Should computer scientists experiment more? *Computer*, 31:32–40, 1998. 4

[48] Steve Vinoski. Serendipitous reuse. *IEEE Internet Computing*, 12(1):84–87, 2008. 26, 48

[49] III Jon W, Orozco Camilo, Goodale James, Wu Chunlei, Batalov Serge, Vickers Tim J, Valafar Faramarz, and Su Andrew I Huss. A gene wiki for community annotation of gene function. *PLoS Biol*, 6(7):e175, 07 2008. 11, 29

Appendix A

Questionary of the Usability Experiment

1. Demographic Information:
    - Name:
    - Age:
    - Genre:
    - Education (Finished degree):
    - Education (Current degree):
2. Experiment experience:
    - Actor / Co-actor:
    - Level of difficulty(Easy, Medium, Hard):
    - Do you think you answer all the questions correctly?
    - Do you think the team work helped to solve the tasks? why?
    - Any comments about the method of the experiment:
    - Working together makes to solve the tasks slower, faster or about the same?
    - Did you find any critical error in the application:
3. Comments or suggestions:

Appendix B

Experiment Reports By Group

B.1 Group 1

B.1.1 Subjects

Table B.1: Information of the individuals - Group 1

Age	Gender	Finished Degree	Current degree	Role	Difficulty
27	M	BSc	MSc	Actor	Medium
28	M	MSc		Co-actor	Medium

B.1.2 Tasks

Task 1

The first 5 fields of this questionnaire were filled without any inconvenience, however they took around 8 min to decide on the name of the protein because they had never explored the *non positional features* panel. They decided to use the name of the first feature, but clearly they were not convinced that this information was correct. For the same reason they didn't find a publication related with the protein, and therefore they left this field empty.

Task 2

They selected the right columns to display without any issue.
About the filtering options, they used the trees properly but they didn't notice that the graphic was updating on the fly, so they refreshed the whole page. By the second time they noticed the automatic update.

Task 3

Error: There was a problem after they created the user: In the writeback panel the user appears as logged but the private functions are still disabled. After a log-out and log-in again the problem was solved. The rest of the groups were alerted about this bug to avoid any wasted time.
Usability Issue: The window to create a user was still open after creating a user, creating confusion to the users.

Task 4

Usability Issue: The information about the meaning of each field of an annotation should be at the side of the field to clarify how to fill in this form.
They struggled to extract the required data from the paper and misinterpreted some of the meaning of the paper, and therefore some fields were filled with the wrong information.
Error: After creating the first feature the graphic did not update this info automatically as expected, it required that all the information was reloaded.

Task 5

It took some time for them to figure out that clicking on a feature opens a popup window with the writeback functionalities.
Suggestion: Create a copy of an existing feature to pre-fill the fields and the user just have to edit to put in the new information.

B.2 Group 2

B.2.1 Subjects

Table B.2: Information of the individuals - Group 2

Age	Gender	Finished Degree	Current degree	Role	Difficulty
27	F	MSc		Co-actor	Medium
30	M	MSc	PhD	Actor	Medium

B.2.2 Tasks

Task 1

Usability Issue: After filling the protein ID field they used the key *enter* to submit the info. However, the system requires a click on the button *go*
They didn't find out that the text over the progress bar was a link to extend the logs of the loaded server, and therefore the information about the server with warnings and the server without features were wrong, however they deduced that the displayed features were recovered from a server that answered.
Usability Issue: Some of the features didn't display the popup window when the mouse was over it.[1]

Task 2

The manipulation tasks were completed with no problems.
Error: When more than one of the filtering trees is used the graphic did not always update coherently.

Task 3

Suggestion: Change the message of user created, because as a warning it makes the user think it is an error.
The user tried to use the *enter* to submit the form and the application just works if the *submit* button is pressed.
The user was created.

[1] See video at the time 10:20

Task 4

Usability Issue: When the suggestion list appeared with the ontology terms the user wanted to used the keyboard to navigate through the suggestions. This component just works with the mouse.
Error: After creating the first feature the graphic did not update this info automatically as expected, it required a reload of all the information.

Task 5

They tried to use the right click to display the writeback options for a feature, however they found out quickly that it was with a left click.

B.3 Group 3

B.3.1 Subjects

Table B.3: Information of the individuals - Group 3

Age	Gender	Finished Degree	Current degree	Role	Difficulty
31	M	MSc	PhD	Actor	Medium
31	M	Postgraduate	MSc	Co-actor	Medium

B.3.2 Tasks

Task 1

The user tried to use the *enter* to submit the form and the application just works if the *go* button is pressed.
The information about the queried servers were mistaken with other data, for example they found a non positional feature called *no output* and they assumed that the owner of that annotation didn't have annotation.

Task 2

The manipulation tasks were completed with no problems.
They used the trees properly, but they thought that it was necessary to submit the query again. In the second try they realized that the graphic was been updated on the fly.

Task 3

User created succesfully.

Task 4

Trying to understand the information required to add a feature, they explored the existing ones and found the kind of information of some of the fields strange. For instance, one of the users said *"Method is Uniprot?", that's weird!* .
Usability Issue: The Type ID field is a problem, because it is required but when the type is filled with a term out of the ontology, the user does not have any coherent value for this field.
Error: After creating the first feature the graphic did not update this info automatically as expected, it required a reload of all the information.

Task 5

Because of the previous errors the user didn't notice that it was required to add more features for this task, and they skipped directly to the next one.

B.4 Group 4

B.4.1 Subjects

Table B.4: Information of the individuals - Group 4

Age	Gender	Finished Degree	Current degree	Role	Difficulty
32	M	BSc	MSc	Actor	Easy
31	M	Postgraduate	PhD	Co-actor	Easy

B.4.2 Tasks

Task 1

They easily found all the answers for this task.

Task 2

They took longer to find the *manipulation options* panel, but afterward they chose the right columns without a problem.
They filtered the features without any problem.

Task 3

User created succesfully.

Task 4

Error: After creating the first feature the graphic did not update this info automatically as expected, it required a reload of all the information. They tried to put some incorrect data to test the system, like the start amino acid after the final one and the application captured the errors on time.

Task 5

They were looking for a way to add a feature from the same track.
Suggestion: Create a duplicate of an existing feature to pre-fill the fields and the user just has to edit to put in the new information. They explored the history of a feature looking to restore one of the features that they deleted.
Suggestion: In the list of deleted features, change the link of *Remove* to *Restore*
Usability Issue: Remove the *rollback* button in the history of a feature for the current version.