A Literature Review for the Problem of Biological Data Versioning

Ben Tagger

Started - 28th July 2005

Contents

List of Figures

1 Introduction

This document aims to provide an idea of the current state of research with regards to biological data management as well as the versioning of biological data. It is not intended that this document provide an exhaustive list of relevant publications. It is my hope to represent the main achievements in the areas appropriate to the area of my intended research. Obviously, the writing of this document will be an ongoing process with relation to the continuing research being carried out. Therefore this document should be updated when needed in order for it to keep in touch with the current states of research (including, hopefully, my own work).

Several relevant areas of research have been identified. These include (but may not be limited to);

- Biological Data Sources
- Biological Data Management
- Generic Data Versioning
- Versioning Frameworks
- Ontologies and The Semantic Web
- Distributed Computing and Data Provenance

It is intended that this document will not be read from cover to cover. Moreover, it will be more useful to simply dip into the areas that may be of interest. I have included small areas of descriptive text; for example, at the beginning of each section. This serves two purposes. Firstly, it helps to tie the document together, rather than simply supplying a list of relevant references. Secondly, it provides with some well-needed practice at document production[1].

[1]For example, I am using LaTeX to create this document, which may account for a rather slower turnaround time.

2 Biological Data Sources

Before discussing aspects of biological data management, it is important to first talk about the various biological data sources currently available to biologists and researchers. One of the issues surrounding biological data management is the heterogeneity of the data and multiplicity of the data sources, so let us investigate this problem here.

2.1 Biological Sequence Formats

Sequence formats are the way in which biological data such as amino acid, protein and DNA sequence is recorded in a computer file. If you wish o submit some data to a data source, it must be in the format that that particular data source is prepared to receive. There are numerous sequence formats in the biological domain, too many to discuss the details and differences in this document. In general, each database will have its own sequence format. Some databases such as GenBank, EMBL and DDBJ (DNA Data Bank of Japan) share similar data formats (albeit with differing headers).

The EBI (European Bioinformatics Institute) ensures that all the information from molecular biology and genomic research is made publicly and freely available in order to promote scientific progress. EBI's databases and tools allow biological information to be stored, integrated, searched, retrieved and analysed. As mentioned above, each available database will require the appropriate tool in order to make successful queries and each database will require data to be submitted in the correct format.

Microarrays are one of the most important breakthroughs in experimental life sciences that allows snapshots to be made of gene expression levels at a genomic stage[2]. Microarray data can be accessed though *Array Express* which is the public repository for microarray based gene expression data. Although many significant results have been derived from microarray studies, one limitation had been the lack of standards for presenting and exchanging such data [1]. MIAME (Minimum Information About a Microarray Experiment) [1] describes the minimum amount of information necessary to ensure that the microarray data can be easily verified and enable the unambiguous interpretation and reproduction.

The effective and efficient delivery of health care required accurate and relevant clinical information. The storage of clinical information has traditionally been limited to paper, text or digitised voice. However, a computer cannot manipulate data in these types of format. Clinical terminologies are also large, complex and diverse with respect the nature of medical information that has been collected over the 130 years of the discipline. There are numerous schemes which have been successful in supporting the collation and comparison of medical information. However, the problem comes when we try to transfer information between schemes. It is also hard to re-use schemes for purposes other than

[2]Taken from http://www.ebi.ac.uk/Databases/microarray.html

which they originally developed and this causes the proliferation of even more schemes. Galen (and the Open source version OpenGalen [2]) provides a formal model of clinical terminology.

Mass spectrometry is a powerful analytical technique that is used to identify unknown compounds and quantify known compounds (even in very minute quantities). It is also used to establish the structure and chemical properties of molecules. Mass spectrometry can be used to sequence biopolymers (such as proteins and oligosaccharides), determine how drugs are used by the body and perform forensic analyses (drug abuse, athlete steroid abuse) among others. Because of the large amounts of information that can be generated by mass spectrometry, computers are essential (not only to control the mass spectrometer), but for spectrum acquisition, storage and presentation. Tools are available for spectral quantitation, interpretation and compound identification via on-line spectral libraries.

2.2 Biological Data Access

According to the EBI website, the general method for data manipulations (at least from the EBI databases) is the following;

- Choose a tool.
- Select a database.
- Enter or upload your sequence.
- Press run and wait for your results.

Choose a tool: The EBI toolbox provides a comprehensive range of tools for bioinformatics. This refers to the method to which the submitted data will be subjected. EBI offers many tools including; similarity and homology search algorithms such as Blast and FASTA; protein function determination services such as CluSTr and GeneQuiz; proteomic services such as UniProt DAS [14]; sequence analysis (identifies similarities between unknown sequences and those sequences that have been functionally identified) such as ClustalW and pairwise-alignment tools; highly accurate structural analysis algorithms such as DALI and MaxSprout.

Select a database: There is a range of molecular biology databases to choose (depending on the type of data that you may wish to use) including; nucleotide sequence, protein sequence, protein function/interactions, gene expression and microarray, gene ontologies, biological structure, scientific literature and text mining and metabolic pathway databases.

Enter or upload the sequence: This will normally/often involve cut and pasting a sequence into a large text window. This sequence should be in the appropriate format (partially formatted sequences will be rejected).

Press run and wait for your results: Your results are either displayed "interactively" or emailed to your email account.

2.3 A Biological Data Problem

This subsection will describe a biological data problem, which highlights an area where data versioning may be of use. We also provide a scenario for determining the context of the problem. In this scenario, experiments are conducted using biological data sources and yield results. These results are, in turn, used as data sources in other experiments to yield further sets of results. This cycle of experimentation continues, presumably until the required set of results are achieved. This can often lead to many cycles of experimentation with complex sets of data sources from many heterogenous sources. For the purposes of simplification and to aid explanation, it is assumed that the experimentation cycle occurs, for want of a better term, in a linear fashion. In other words, the results of one cycle form the entire data source for the next cycle with no other inputs.

Figure 1 illustrates this cycle of experimentation.

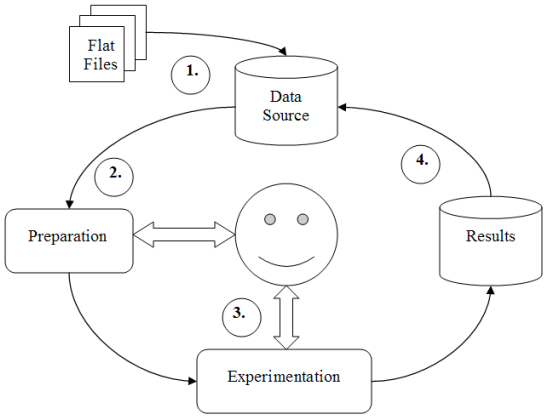

Figure 1: The cycle of experimentation and the re-use of results.

1. Importing biological data. Of all the scientific disciplines, biology has one of the most complex information-structures (concepts, data types and algorithms) [3]. Its richness and diversity provides many challenges for biological sciences, computational sciences and information technology. There exists a collossal amount of biological data which is stored in a variety of formats in a multitude of heterogenous systems. Accessing the relevant data, combining data sources and coping with their distribution and heterogeneity is a tremendously difficult task.

Data integration of geographically dispersed, heterogenous, complex biological databases is a key research area [15]. One of the principal issues of data integration is the data format. Many current biological databases provide data in flat files which are poor data

exchange formats. Worse still, each biological database has a different format. This makes the integration of such datasets difficult and time-consuming. In the example illustrated in figure 1, the flat file containing the biological data will be imported (usually downloaded over the Internet) and placed into whatever structure the bioinformaticien has devised for the purpose.

2. Preparation of the data. It is unlikely that, after importing the data, it will be ready for experimental use. It is also unlikely, although possible, that the data will be in the required format for the experimentation. Therefore, the data must be *wrapped*. *Wrapping* a data source refers to getting data from somewhere and translating it into a common integrated format [4]. The intended experimentation will seldom use the entire downloaded data source (this flat file including all the possible information), but rather a selected dataset. Therefore, the bioinformaticien may wish to manually select the dataset before starting the experiment. This can result in a considerable saving of experiment time. Manipulating and wrapping the data source can be a very timely process and may also depend on the nature of the experimentation to be done. It also involves considerable input from the scientist.

3. Experimentation. After the data has been prepared and the experiment is ready to be carried out, the scientist still needs to do a few more things. Firstly, the data source must be identified. Secondly, the method of experimentation itself must also be selected. This may be a third-party tool or a user-defined software program. Whatever the nature of the method, it is likely that the user will also want to specify some parameters. The parameters may relate to the form of the datasets being used in the experiment, the structure of the required output (results) or they may relate specifically to the bioinformaticien's experimental hypothesis. In any case, the state of the parameters will have an effect on the experiment and, therefore, the results.

4. Renewing the data source. When the experiment is complete, the results are placed in a data repository where they become available for further experimentation. This cycle will continue until the required results are obtained and this may result in many cycles.

The entire process can take a long time and that, itself, is a problem that requires attention. A more serious issue is the update to the initial biological data. The large biological data repositories (from which the data is taken as the initial data source) currently have no way of dynamically reflecting changes, updates, additions and deletions. Therefore, they release versions of the data (in flat files) every so often[3].

Clearly, it is preferable for the bioinformaticien's results to reflect the latest changes from the initial flat file data source. However, with the current methodology, every result is dependent on the initial data source. In order to get to the same stage (as before the new data version was released), the bioinformaticien must perform each experiment as before, only this time using the updated dataset. This will result in their having to re-

[3]The length of time between versions varies between database providers.

do all previously conducted experiments, which also depends on their having accurately recorded the experimentation already done.

After identifying the stakeholders, one of the first stages of a requirements analysis is *scenario production*. Scenarios can best be viewed as telling a story about an activity from the view point of a particular participant [**?**]. They provide a description of a particular transaction. This description can be used to define requirements and explain how the participant would expect some particular activity to behave.

Scenarios can aid the discovery of user-oriented requirements (what the user will expect or require the system to achieve) as well as identify external influences on the system. The Scenario can be thought of as an ideal world example. The following paragraph provides a scenario whereby the user interacts with a system aimed to address the problem described in section two.

The user downloads the required biological data in a flat file format from the appropriate web site. However, there is a lot of redundant data in the file (data the user has no intention of using), so the user designs a simple database query to select only the tuples of interest. Since this is the first time the user has downloaded a data file with this format, he must write a simple java method to correctly parse the data. Once he has successfully selected the right data and has the means to use it, the user then selects the statistical test he wants to execute. For the experiment, he selects the data, the method and the parameters. He runs the experiment, which takes about two hours, and he gets a table of results, which he saves in a database. Over the next three months, he conducts series of further experiments, based on the results from the first experiment (described above). Each experiment will have certainly involved either a different data source, method or set of parameters with differing requirements for the preparation of the data. The web site from where the data file was originally downloaded reports that they have an updated version of the flat file available. The user downloads the file and saves it to the system as a new version. The system reports to the user that the data file has been modified in a way that changes some of the results of his previously conducted experiments. The system asks the user whether he would like to update his results and informs him that it will take approximately 16 hours. He confirms that he does want to update the results but, in order to prevent it from disrupting his work more than is necessary, schedules the update at six o'clock that evening. He returns the next day to find the update still running. At eleven o'clock that morning, the update finishes and the user is pleased to find his results up to date and reflecting whatever changes had been made to the original source data file.

3 Biological Data Management

Over the past twenty or so years, biology and biological research has largely been dominated by what has become known as the genomic era. It was believed that the mapping

of the human genome would herald a new world of discovery for the life sciences. Unfortunately, little (by comparison) has come to fruition due to the lack of understanding of the genome.

A vast amount of genomic and proteomic data has been collected but there is still a lot to be done to allow its full understanding. The vastness of the data represents a part of the problem. There has been a significant amount of research on the problems attached to biological data and its management and I will endeavour to represent the most significant advances in this chapter.

Of all the scientific disciplines, biology has one of the most complex information-structures (concepts, data types and algorithms) [3]. Its richness and diversity provides many challenges for biological sciences, computational sciences and information technology. For the purposes of this document and with regard to my research, the term 'biological data' refers to data that is in digital format.

3.1 The Problems with Biological Data

As is true of most scientific data, biological data possess the following characteristics.

- Complexity
- Incompleteness
- Error-prone
- In very high demand for a variety of applications.

To add to these less than ideal characteristics, there exists a collossal amount of biological data which is stored in a variety of formats in a multitude of heterogenous systems. Scientists need an integrated view of these heterogenous data sources with advanced data accessing, analysing and visualisation tools [4]. Accessing the relevant data, combining data sources and coping with their distribution and heterogeneity is a tremendously difficult task. The continuing data growth will lead to an increasing need for large-scale data management. Biological discovery depends, to a large extent, on the presence of a clean, up-to-date and well-organised dataset [6].

The aspects of biological data management are almost as diverse as the data itself. In the following sections, the most appropriate and accredited areas are explored. The areas of biological data management that are to be addressed in this section are:

- Data retrieval and access
- Data annotation systems
- Data integration problems and tools
- Data cleaning

3.2 Data Retrieval and Access

There are a number of data sources from which scientists and researchers may wish to retrieve and access biological data, including[4]; GenBank, RefSeq, UniProt, Human Protein Reference Database (HPRD), Biomolecular Interaction Network Database (BIND), Database of Interacting Proteins (DIP), Molecular Interactions Database (MINT), IntAct, NCBI Taxonomy, Gene Ontology (GO), LocusLink, Entrez Gene, HomoloGene and many, many more.

In order to understand the problems associated with biological data management, it is necessary to first understand the current practice of this process. Currently, biological data management is largely conducted within specialist bioinformatic databases (data warehouses). Specialist bioinformatic databases contain detailed information (such as functional and structural properties, expert-enriched information) that is essential for the in-depth analysis of the relevant data [6]. This level of detail is usually lacking in the primary databases such as SwissProt or GenBank. These specialist databases generally exhibit the following properties/qualities;

- Increased detail of annotation
- Cleaner data
- Integrated data from multiple sources
- (Often)Integrated searches and analysis tools

Schönback et al [7]defines such a biological data warehouse as a subject-oriented, integrated, non-volatile, expert-interpreted collection of data in support of biological data analyses and knowledge discovery.

According to [6], the key steps to constructing a data warehouse (a Specialised Sequence data warehouse in the case of [6]) include the extraction of raw data from multiple sources and transformation to the data warehouse format (*data integration*), removing errors and discrepancies (*data cleaning*), enriching the data with information from relevant literature or from experimental results (*data annotation*) and, finally, the construction of a new database[5].

BioWare, A framework for the construction of a biological data warehouse, is provided by [6]. As opposed to many SSDWs (Specialised Sequence Data Warehouse(s)), the users of BioWare do not require any specific programming or database expertise in order to retrieve, annotate and publish their data in the form of a searchable SSDW. However, BioWare does have some limitations. Given that the SSDW is designed to accessed via HTTP, a bottleneck is created with excessive downloading from public data sources. The current system, therefore, holds only 1000 entries. The data is still held in flat files, which impedes the speed of data access.

[4]Taken from [5]

[5]Incidentally many of these operations are included in this Biological Data Management section.

The EnsMart system [8] purports to provide a generic data warehousing system for fast and flexible access to biological datasets as well as integration with third-party data and tools. EnsMart is a system that is capable of organising data from individual databases into one query-optimised system. The generic nature of EnsMart allows the integration of data in a flexible, efficient, unified and domain-independent manner [8]. EnsMart is a self-contained addition to Ensembl software and data, providing access to commonly used parts of the genome data. One of the noted benefits of the EnsMart system is the ability to engineer your own version (with a little informatics expertise). This is useful is you wish to use data and queries that are not explicitly offered by the website. This will also overcome the problem of slowness or 'out-of-service' at www.ensembl.org when mining data in large quantities. EnsMart is reported to offer new levels of access and integration for the use of large quantities of genome data [9].

3.3 Data Annotation Systems

Annotations are features on the genome derived through the transformation of raw genomic sequences into information by integrating computational tools, auxiliary biological data and biological knowledge[6]. With the completion of the human genome and genome sequences being readily available, the task of manual annotation has become a large priority and increases to be so, as we look into the future [10].

In the beginning, bioinformatics data was stored in flat files as ASCII characters. This was sufficient as the number of known proteins were small and there was little thought of genomic data. With the advent of rapid gene sequencing, it was established that techniques such as dynamic programming would be needed to cope with the exponential growth of data [11].

Ensembl [12] began as a project to automatically annotate vertebrate genomes. This basically involves running many commonly-used bioinformatics tools and combining the outputted data to produce genome-wide data sets such as protein-coding genes, genome-genome alignments and other useful information. Obviously, there is a considerable amount of processor power needed to produce the annotation for the vast amount of biological data. It is not feasible to compute this kind of problem on a single-CPU, but rather employ a distributed machine infrastructure.

Currently, most gene annotation is curated by centralised groups and little effort has been made to share annotations between multiple groups.The Distributed Annotation System (DAS) allows sequence annotations to be decentralised among multiple annotators and integrated by client-side software [13]. DAS has been successfully used to integrate protein annotations [14].

[6]Taken from ISMB '99 Tutorial 3: The challenge of annotating a complete eukaryotic genome: A case study in Drosophila melanogaster

3.4 Data Integration Problems and Tools

Data integration of geographically dispersed, heterogenous, complex biological databases is a key research area [15]. One of the principle issues of data integration is the data format. Ideally, a simple, self-describing format is best. However, many current biological databases provide data in flat files which are poor data exchange formats. Worse still, each biological database has a different format. Bio2X [15] is a system that gets around this problem by converting the flat file data into highly hierarchical XML data using rule-based machine learning techniques.

Data integration consists of wrapping data sources and either loading the retrieved data into a data warehouse or returning it to the user. *Wrapping* a data source means getting data from somewhere and translating it into a common integrated format [4]. There have been many systems designed for the specific purpose of biological data integration (too many to mention here). None of these approaches provides a seamless integration that permits the use of metadata about source content and access cost to permit the optimisation of the evaluation of the queries [4].

A wide variety of biological experimental techniques are available from the classic methods to high-throughput techniques such as gene expression microarrays. In the future, more disparate methods will be developed, which will continually push integrative technologies and continue to drive research in this area. MAGIC (Multisource Association of Genes by Integration of Clusters) is a general framework that uses formal Bayesian reasoning to integrate heterogenous types of high-throughput biological data [16].

3.5 Data Cleaning

As the amount of biological data increases and becomes more pervasive, there arises various issues surrounding data quality, data legacy, data uniformity and data duplication. *Data cleaning* is the process by which biological data is corrected and standardised. However, due to the complex and diverse nature of the data, the problem of improving the data quality is non-trivial [17]. If quality of the data is not maintained, then the processes and operations that rely on this data will either fail or (arguably worse) provide skewed results.

BIO-AJAX [17] is a toolkit that provides an extensible framework with various operations to address data quality issues through data cleaning within biological data repositories.

3.6 Metadata

Metadata literally means "data about data" and is a term that is understood in different ways by the diverse professional communities that design, create, describe, preserve and

use information systems and resources [18]. In other words, metadata is structured information that describes, explains or locates, or otherwise makes it easier to retrieve, use or manage an information resource.

Metadata is key to ensuring that resources will survive and continue to be accessible in the future. It is generally understood that there are three main types of metadata[7].

1. *Descriptive* metadata describes a resource for the purpose of discovery and identification. It may include descriptive elements such as title, author, data, keywords, etc.
2. *Structural* metadata provides information as to how the data is structured. I.e. how pages are ordered into chapters.
3. *Administrative* metadata provides information to help manage the resource, such as when and how it was created, file type and other technical information.

According to [18], in an environment where a user can gain unmediated access to information objects over a network, metadata:

- certifies the authenticity and degree of completeness of the content.
- establishes and documents the context of the content.
- identifies and exploits the structural relationships that exist between and within information objects.
- provides a range of intellectual access points for an increasingly diverse range of users.
- provides some of the information an information professional might have provided in a physical reference or research setting.

Metadata plays an important role in providing semantic meaning to resources and promotes interoperability (the ability of multiple systems with different hardware and software platforms to exchange data with minimal loss of content and functionality).

The existence of information in digital form has heightened interest in the ability to create multiple and variant versions of those objects. Metadata is used to link multiple versions and capture what is the same and what is different about each version. The metadata must also be able to distinguish what is qualitatively different between variant digitised versions and the hard copy original or parent object [18].

[7] This is also referred to as content, context and strucuter [18].

4 Generic Data Versioning

Although I have found a substantial amount of research and publications concerned with the management and problems arising from the management of biological data, I have been unable to find anything on biological data versioning specifically. This is, of course, good news for me as it is the area that I am working on. The disadvantage is that there isn't anything specifically relevant to start with.

I have, therefore, decided to look at generic data versioning with a view to investigating the feasibility of applying existing data versioning techniques within the biological data domain. There has been a great deal of research done on data versioning. I intend, however, to only cover those areas that I believe are appropriate to biological data.

4.1 Uses of Versioning

The benefits of versioning can be grouped into three categories [19].

- Recovery from user mistakes
- Recovery from system corruption
- Analysis of historical change

The first two items above, although important in their own right, are not specifically applicable to the problem of biological data versioning. The analysis of historical change is important, however, as it is specifically the history of the data in which we are interested. Analysis of the history can help us answer questions about how a file reached a certain state [19]. For example, [20] and [23] keep a complete record of committed changes to specific files.

The analysis of historical change has been used largely to determine when and if files have been corrupted or changed due to malicious use. It can be used to help identify the 'movements' of an unwanted visitor to a filestore and then potentially reverse them or at the least, estimate the damage caused [24].

4.2 Methods of Versioning

There have been many approaches to the versioning of files and data and I shall endeavour to describe some of them here. Again, this section is not intended to be exhaustive, moreover it seeks to identify the major contributions. It should be noted that [25] provides a detailed background section, which has been used (albeit not exclusively) for this section.

There were several early file systems that provided versioning, such as the Cedar File System (CFS) [27] and 3D File System (3DFS) [26]. However, these systems were

not transparent. That is to say, users had to manually create versions using special commands and tools. With CFS, users had to change a copy of the file on their local system. A file was only versioned when it was transferred to the remote server.

CVS [20] is also not transparent as users have to use specific commands to control the versioning of their files. CVS utilises a client-server architecture, whereby a client connects to the server in order to check-out a complete copy of the project, work on this copy and then later check-in their changes. CVS has become quite popular in the open-source world. Rational ClearCase [28] is another versioning tool, but requires specific administration and is also expensive.

There are also various snapshotting tools available such as WAFL [29] and Ext3cow [30]. With the snapshot method, incremental or whole snapshots of the file system are made periodically. When required, the entire file system can be restored from any recorded point in history. However, snapshotting makes no allowances for how often a file may be changed, it simply works on a discrete time period of data capture. The consequence of this is that files which do not change regularly will be continually 'snapshotted' regardless. Conversely a file that is being regularly updated may have updates that are not captured with the snapshot method. This suggests that it may be inappropriate for biological data versioning. Also, entire snapshots must be purged if the space is to be reclaimed.

Versioning employing the copy-on-write technique is used by Tops-20 [31], VMS [32], the Elephant File System [33] and CVFS [19, 34]. An ideal situation would be to have versioning automatically integrated into the operating system. However, this has yet to be implemented.

4.3 Current Object Versioning Methodology

One of the main issues in biological data versioning is how to deal with the sheer vastness of the data. Conventional versioning systems do not efficiently record large numbers of versions. Given the large amount of biological data, the prospect of storing multiple versions of the data (as well as the associated metadata) is an unpleasant one.

This subsection aims to explore the methodology behind current versioning techniques and describe some of the differences between various approaches. According to [35], there are two main methods for dealing with object versioning. The first technique stores versions of a particular object as complete objects and is referred to as *complete versioning*. This approach is fairly easy to implement, although the waste of storage space becomes more detrimental as the number of versions increases. The second technique stores a single object and each version is maintained as a difference between the current version and the previous version. This approach is more difficult to implement but it is more suitable for representing data that may be subject to continuous and dynamic changes [35].

Using this approach, changes in objects are handled using a version management system. Each version reflects a change in the object's attributes and/or behaviour. Subsequent changes in the object (versions) will generate related dynamic attributes and temporal links to the updated versions. This type of version management reduces the need for large storage space[8], since the current object is only stored once in its entirety [35].

Linear versioning is a technique where one version is stored as a complete object and the rest of the versions are represented as differences between the versions. This approach is based on one-to-one versioning. That is to say, that each parent or base object will have only one child or derived object. Linear versioning can be classified into two versioning strategies [35]. The first allows the current version to be calculated from the initial base version (with the addition of the subsequent versions representing the changes over time) and is referred to as *forward oriented versioning*. The second strategy stores the current version as the base version and previous versions are calculated as differences to the current version. This is known as backward oriented versioning [36].

Which method to choose varies greatly on the type of manipulation that the data is to be subject to. Forward oriented versioning takes longer to retrieve the current object (differences must be added to the base version to get the current version). However, it takes less time to retrieve earlier versions. Backward oriented versioning provides faster access for the newest versions. As a result, this strategy is usually more suitable for most applications [35].

Branching is a technique where one version is stored as a complete object and all other versions are stored as differences between that version and other versions. As all versions are just one 'difference' away from the current version, the *branch forward versioning* strategy provides the same access time for all versions.

What are the strategies for working out the differences between versions?

It is important to look at all versioning strategies (including combinations of strategies) in accordance with the nature of the problem that you are trying to solve.

4.4 The Use of Metadata in Versioning

Metadata is a description of the content, quality, lineage, contact, condition and other characteristics of data[9]. Biological metadata works in the same way. Information about the data, including content, quality, and condition makes the data more easily accessible to scientists and researchers. Unfortunately, conventional versioning systems do not efficiently record large numbers of versions. In particular, versioned metadata can

[8]Storage is definitely a serious consideration given the vast amount of biological data that may be effected by version management.

[9]According to the National Biological Information Infrastructure website - http://www.nbii.gov/datainfo/metadata/

consume as much space as the data itself [19]. Two space-efficient metadata structures for versioning file systems are examined in [19].

4.5 Schema Evolution and Temporal Versioning

A schema describes the structure of the data that is being stored. It is possible to evolve schema in response to the changing structure of the data. Schema evolution is a way of dealing with the changes that the data warehouse (DW) schema undergoes over time in response to evolving business requirements [37, 38].

4.6 Examples of Successful Data Versioning

The purpose of this section is not to look at versioning methodology as a whole, but rather look at specific examples of where versioning has worked in areas unrelated to the life sciences. Of particular importance are areas that seek to version data that is similar in structure to biological data. This section attempts to identify some specific areas where data has been successfully versioned and section four aims to explain how these techniques may (or may not) be applied to versioning biological data. It should be noted that this section is not intended to provide an exhaustive list of data versioning successes, but rather to present a few examples to demonstrate some different techniques of data versioning.

Business data is largely focussed on the most current version of the data. Previous versions are kept as a way of providing historical data for analysis. This is useful for documentary analysis and visualising business trends (showing improvement or decline), but is not as widely used as the current data version.

During the life cycle of a piece of software, different versions may be developed depending on the state of development and the purpose of the intended release. In order to effectively support software development, these versions as well as the relationships between them, must be stored [21]. These differing versions may be due to the development of the software (i.e. the adding of functionality). However, a specific set of versions may be needed to fulfil the requirements for a certain release of the software. For example, a shareware release of a software item may require a previous set of versions (with limited functionality). A beta release may require the latest set of versions, irrespective of the level of testing undergone. Software versioning is often conducted with a file versioning system, such as CVS [20]. CVS utilises a client-server architecture, whereby a client connects to the server in order to check-out a complete copy of the project, work on the copy and then later check-in their changes. Basic software versioning techniques typically provide versioning of large granularity objects (at the file level), whereas it may be more useful and appropriate to version at a finer granularity level (at the object and instance level).

An interesting example of successful data versioning in a scientific domain is presented by Barkstrom [22]. According to [22], large-scale scientific data production for NASA's Earth observing satellite instruments involves the production of a very vast amount of data from a very wide variety of data sources[10].It is noted that while software versioning requires tracking changes principally in the source code, versioning of the data requires the tracking of changes in the source code, the data sources and the algorithm coefficients (parameters). Barkstrom [22] notes that changes in any of these items can induce scientifically important changes in the data.

4.7 Other Versioning Technologies

Normally, versioning functionality has been implemented by high-level applications or at the filesystem level. [39] proposes a technique that pushes the versioning functionality closer to the disk by taking advantage of modern, block-level storage devices. Versioning at the block-level has several advantages over versioning at the filesystem or application levels. Firstly, it provides a higher level of transparency and is completely filesystem independent [39]. It also reduces complexity in the higher layers, in particular the filesystem and storage management applications [40]. Thirdly, it offloads expensive host-processing overheads to the disk subsystem, thus, increasing the overall scalability of the system [40].

However, there are disadvantages to versioning at the block level. Having offloaded complexity from one level to another, it is still picked up somewhere and the ramifications of this transference is unclear. The consistency of the data may be affected with the use of the same volume as the filesystem. But perhaps the most relevant disadvantage with respect to biological data versioning is the problem of versioning granularity. Since the data versioning is occurring at a lower system layer, information about the content of the data is unavailable. Therefore, access is only available to full volumes as opposed to individual files.

5 Versioning Frameworks

Various versioning frameworks such as the use of ontologies and RDF for the semantic web are described later in this document. The aim of this section is to include any other generic versioning strategies and frameworks that may not be specifically relevant to other areas. Therefore the content in this section may be quite limited (as the content will be included elsewhere).

Versioning systems are generally required to manage data objects that change frequently over time [41]. The objective is to achieve the ability of representing intermediate stages

[10]In fact, Barkstrom [22] quotes the production values at tens of thousands of files per day from tens or hundreds of different data sources.

in the object's evolution paths as *versions*. [42] defines a version a version as a semantically meaningful snapshot of a design object (an artifact) at a point in time. One objective of versioning is to allow users access to these objects at previous points in time.

[43] investigates the use of versioning systems to facilitate the return to previous stages of work if a particular path of the design process doesn't work out. The versioning of an engineering process enables the reusability of a design path since a new design version can be derived from a previous design version. Object-oriented database management systems (OODBMS) can employ versioning at any level of granularity (e.g. schema, class, object). [43] presents a framework for checking each object version for data correctness and consistency.

6 Ontologies and The Semantic Web

There has been a great deal of research conducted in the areas of ontologies and the semantic web, far too much to be exhaustively investigated here. I hope to provide a concise summary of the key points with respect to the problem of Biological data versioning.

An ontology is;

> An ontology is a formal, explicit specification of a shared conceptualization. Conceptualization refers to an abstract model of phenomena in the world by having identified the relevant concepts of those phenomena. Explicit means that the type of concepts used, and the constraints on their use are explicitly defined. Formal refers to the fact that the ontology should be machine readable. Shared reflects that ontology should capture consensual knowledge accepted by the communities. [44]

6.1 Ontologies

There is a large amount of heterogeneous biological data currently available to scientists. Unfortunately, due to its heterogeneity and the widespread proliferation of biological databases, the analysis and integration of this data represents a significant problem. Biological databases are inherently distributed because the specialised biological expertise that is required for data capture is spread around the globe at the sites where the data originate. To make the best use of this data, different kinds of information must be integrated in a way that makes sense to biologists.

Biologists currently waste a great deal of time searching for available information in various areas of research. This is further exacerbated by the wide variations of terminology used by different researchers at different times. An ontology provides a common vocabulary to support the sharing and reuse of knowledge.

The OBO ontology library [45] is a repository of controlled vocabularies, which has been developed in order to provide for improved communication across different biological and medical domains. The Gene Ontology project

The Gene Ontology (GO) project provides structured, controlled vocabularies and classifications that cover several domains of molecular and cellular biology. The project is driven and maintained by the Gene Ontology Consortium. Members of this consortium continually work collectively and, with the help of domain experts, seek to maintain, expand and update the GO vocabularies. Collaborations of this nature are difficult to maintain due to geography, misunderstandings and the length of time it takes to get anything done.

6.1.1 GO Consortium

The Gene Ontology (GO) project is a collaborative effort that aims to address two aspects of information integration:

- Providing consistent descriptors for gene products in different databases.
- Providing a classification standard for sequences and sequence features.

The project began in 1998 as a collaboration between three model organism databases: FlyBase (Drosophila), the Saccharomyces Genome Database (SGD) and the Mouse Genome Database (MGD). Since then, the GO Consortium has grown to include many databases . The benefits of using GO increase as the number of participating databases increases. The GO project has grown enormously and is now a clearly defined model for numerous other biological ontology projects that aim to achieve similar results.

6.1.2 Ontology Methodology

Barry Smith is one of the leading characters in the development and analysis of Ontologies. His studies are not limited to biological Ontologies and have also addressed much of the methodology behind Ontologies. His latest publication [46] addresses the issue of relationships between biological Ontologies. [46] investigates the importance of vigorous, formal relations to ensure the maximally reliable curation of each single ontology while at the same time guaranteeing maximal leverage in building a solid base for life-science knowledge integration.

In [46], Smith also investigates some of the shortcomings of biomedical Ontologies at the moment. He also addresses the problem of under-specification of formal Ontologies in [47]. He has conducted study on data integration with Ontologies [48] and has also been involved in the development of tools to conduct such data integration [49].

6.1.3 Ontology Versioning on the Semantic Web

Ontologies are considered the basis building blocks of the Semantic Web as they allow machine supported data interpretation reducing human involvement in data and process integration [55]. Ontologies provide a reusable piece of knowledge about a specific domain. However, these pieces of knowledge are often not static, but evolve over time [50]. The evolution of ontologies causes operability problems, which hamper the effective reuse. Given that these changes occur and given that they are occurring within a constantly changing, decentralised and uncontrolled environment like the web, support to handle these changes is needed. This is especially prudent with respect to the semantic web, where computers will be using the data (Humans are more likely (but still unlikely, however) to spot erroneous data due to unexpected changes).

Also, consider the nature, complexity and quantity of dependencies that exist between data sources, applications and ontologies. Changes in one area will have far-reaching effects [50]. [50] goes on to provide a framework for the versioning of ontologies and seems a very interesting example of the versioning of something that is 'a bit tricky'.

Traditional versioning systems (e.g. for the use of text and code) enable users to compare versions, examine changes, and accept or reject changes. However, an ontology versioning system must compare and present *structural* changes rather than changes in the text representation of the ontology. [51] presents the PROMPTDIFF ontology-versioning environment, which reportedly addresses these challenges. PROMPTDIFF includes an efficient version-comparison algorithm that produces a structural diff between ontologies.

6.2 The Semantic Web

The next generation of the Web is often characterised as the *Semantic Web*. The semantic web refers to a system whereby information will no longer be easily-accessible for human readers, but also for processing by machines, enabling intelligent information services, personalised Web-sites and semantically empowered search-engines [53]. The semantic web requires interoperability on the semantic level. *Semantic interoperability* requires standards not only for the syntactic form of documents, but also for the semantic information.

> "The Semantic Web will enable intelligent services such as information brokers, search agents, information filters etc. Such intelligent services on the Knowledgeable Web should surpass the currently available version of these services, which are limited in their functionality, and (most importantly), only work as stand-alone services that do not interoperate." [53]

6.2.1 XML and RDF

XML (Extensible Markup Language) and RDF (Resource Description Framework) were the current standards for establishing semantic interoperability on the Web. However, XML only describes document structure. RDF better facilitates interoperation because it provides a data model that can be extended to address sophisticated ontology representation techniques [53].

XML is intended as a markup-language for arbitrary document structure. An XML document consists of a properly nested set of open and close tags, where each tag can have a number of attribute-value pairs. One of the important aspects of XML is that the vocabulary is not set, but rather can be defined per application of XML. The following example XML shows a part of a defined ontology.

Figure 2: The XML representation of part of a defined ontology

XML is foremost a means for defining grammars. Any XML document whose nested tags form a balanced tree is a well-formed XML document. A DTD (Document Type Definition) specifies the allowed combinations and nesting of tag-names, attribute-names, etc. using a grammar formalism. The purpose of a DTD is to define the legal building blocks of an XML document. It defines the document structure with a list of legal elements. A DTD can be declared inline in your XML document, or as an external reference.

RDF was designed by the W3C in order to provide a standardised definition and use of metadata. *Resource Description Framework*, as its name implies, it is a framework for describing and interchanging metadata. A *Resource* is anything that can have a URI (Universal Resource Indicator) such as a web page or an element of an XML document. A *PropertyType* is a resource that has a name and can be used as a property. A *Property* is the combination of a Resource, a PropertyType and a value. A typical example of a Property might be: "http://www.cs.ucl.ac.uk/people/B.Tagger.html, Author, Ben Tagger." This would be represented as

```
<RDF:Description href='http://www.cs.ucl.ac.uk/people/B.Tagger.html'>
<Author>Ben Tagger</Author></RDF:Description>
```

RDF is carefully designed to have the following characteristics[11]:

[11] According to http://www.xml.com/pub/a/98/06/rdf.html

Independence: Anyone can use RDF to invent their own types of metadata. The PropertyType can be anything and is not domain-specific.

Interchange: RDF properties can be converted into XML.

Scalability: As RDF properties are essentially made up of three distinct parts (see above), they are easy to handle and look up, even in large numbers. It is important that metadata is efficiently captured. With the large amount of data, it is important that the overhead caused by metadata is as minimal as possible.

In particular, XML falls down on the issue of scalability. Firstly, the order in which elements appear in an XML document is significant and can change the meaning of the document. When it comes to semantic interoperability, XML has disadvantages. Since XML only deals with the structure of the document, there is no way of recognising or extracting semantic meaning from a particular domain of interest.

6.2.2 LSIDs

Web standards, including RDF, make use of a family of identifiers collectively referred to as Universal Resource Identifiers (URIs). Members of this family include URNs (Unique Resource Name) and Universal Resource Locators (URLs). The latter is used for both specifying the identity and location of a resource.

Life Science Identiers (LSIDs) promise to uniquely and consistently identify data resources in life science, and provide a resolution protocol for the retrieval of that data and any associated metadata [54]. They have been initially developed under the umbrella of the I3C[12] and are now undergoing formal standardisation by OMG[13].

6.2.3 A Semantic Web Use Case for Biological Data Integration

The problem of interoperability between data sources is challenging for the following reasons (According to [52]):

1. The identification of interoperable data, let alone its functionality, is a difficult problem due to the lack of widely-accepted standards for describing the web sites and their contents.
2. Different resources make their data available in heterogenous formats. Some data may be available in HTML format (readable by web browsers, but other data may only be made available in text (e.g. tab or comma-delimited) or binary formats (e.g. images). Such heterogeneity makes interoperability very, very difficult.

[12]Interoperable Informatics Infrastructure Consortium - http://www.i3c.org
[13]Object Management Group - http://www.omg.org

3. Data interoperability requires both syntactic and semantic translation. Both of these types of translation are hindered by the lack of standard data models, formats and ontologies/vocabulary.

YeastHub [52] demonstrates the use of RDF metadata/data standards and RDF-based technologies to facilitate integration of diverse types of genome data provided my multiple web resources in heterogenous formats. Within YeastHub, a data warehouse has been constructed to store and query a variety of yeast genome data obtained from various sources. The application allows the user to register a dataset and convert it into RDF format. The datasets can then be queried. The goal of the application is to facilitate the publishing of life sciences data using the RDF format. The benefits of using the RDF format are [52]:

1. It standardises data representation, manipulation and integration using graph modelling methods.
2. It allows the exploration of RDF technologies and RDF query languages in order to integrate a wide variety of biological data.
3. It facilitates development and utilisation of standard ontologies to promote semantic interoperability between bioinformatic web services.
4. It fosters a fruitful collaboration between the artificial intelligence and life sciences research communities.

However, feedback on YeastHub has been found to be less than glowing.

> "Eagerly, I tested the data base but was a little let down: While I see the obvious benefits, many small problems appear, which taken together, make the system not really very helpful to the average bioinformatics user, let alone a biologist in the the yeast community.
>
> The lack of descriptions of formats of the data sources makes it tough to create queries and I did not really get past trivial results. I also dearly missed capabilities to browse the data sets. However, it appears as if these technologies will become more and more important and the old tab separated tables hopefully disappear one fine day."[14]

6.2.4 Semantic Web Services and Processes

The main idea behind web services is to capture an organisation's functionality within an appropriate interface and advertise it online as a web service [55]. Web services promise universal interoperability and integration, but the key to achieving this relies on the efficiency of discovery of appropriate web services and composing them to build

[14] From "A Bioinformatics Blog" - http://binf.twoday.net/topics/Databases/

complex processes (*Semantic Web service integration*). This potential for businesses and organisations to be able to interact with each other on the fly is very appealing.

Within the web service domain, semantics can be classified into the following categories (According to [55]):

Functional Semantics: One of the most important aspects of Web services is their functionality. If the functionality is not correctly established, then the power of the Web service is neutralised. Web service functionality is established by the examination of the service's inputs and outputs [56]. This approach may not always provide an appropriate set of services. This can be remedied by having a specialised ontology for recognising functionality. Therefore, the intended functionality of each Web service can be represented as annotations using this ontology.

Data Semantics: Details of the input and output of a particular Web service are represented in the signature of the operations in a specification file. To effectively perform discovery of services, the semantics of the input/output data has to be taken into account. If the input/output data is annotated using an ontology, then the data semantics can be used in reference against the required data semantics.

QoS Semantics: After discovering the most appropriate Web service (or list of services), the next step is to select the most suitable service. Services are measured in terms of their QoS (Quality of Service) and this requires the management of QoS metrics for the services.

Execution Semantics: This encompasses the ideas of message sequence, conversation pattern of Web service execution, flow of actions, preconditions and the effects of Web service invocation. Using execution semantics can help in dynamically finding services that will match not only the functional requirements, but also the operational requirements (such as long-running interactions, complexity of operation).

These different types of semantics can be used to represent the capabilities, requirements, effects and execution of a Web service.

7 Distributed Computing

Distributed computing can be thought of as an environment where you can harness idle CPU cycles and storage space of tens, hundreds or even thousands of networked systems to work together on a particularly processing-intensive problem. The main goal of a distributed system is to connect users and resources in a transparent, open and scalable way.

Transparency: Any distributed system should hide its distributed nature from its users, appearing and functioning as a normal centralised system.

Openness: A property of distributed systems that measures the extent to which it offers a standardised interface that allows it to be extended and scaled. It is clear that a system that easily allows more computing entities to be plugged into it and more features to be easily added to it has an advantage over a perfectly closed and self-contained system.

Scalability: A scalable system is one that can easily be altered to accommodate changes in the amount of users, resources and computing entities affected to it.

7.1 Distributed Computing Architecture

There are various hardware and software architectures that exist for distributed computing. At a lower level, it is necessary to connect multiple CPUs with some sort of network, regardless of the nature of the network. At a higher level, it is necessary to interconnect processes running on different CPUs with some sort of communication system.

The following list describes the various types of architectures that can be employed for a distributed computing system [57].

Client-server: Smart client code contacts the server for data, then formats and displays it to the user. input at the client is committed back to the server when it represents a permanent change.

3-tier architecture: Three tier systems move the client intelligence to a middle tier so that stateless clients can be used. This simplifies application deployment. Most web applications are 3-Tier.

N-tier architecture: N-Tier refers typically to web applications which further forward their requests to other enterprise services. This type of application is the one most responsible for the success of application servers.

Tightly coupled (clustered): refers typically to a set of highly integrated machines that run the same process in parallel, subdividing the task in parts that are made individually by each one, and then put back together to make the final result.

Peer-to-peer: An architecture where there is no special machine or machines that provide a service or manage the network resources. Instead all responsibilities are uniformly divided among all machines, known as peers.

Service oriented: Where system is organized as a set of highly reusable services that could be offered through a standardized interfaces.

Mobile code: Based on the architecture principle of moving processing closest to source of data.

Replicated repository: Where repository is replicated amongst distributed system to support online/offline processing provided this lag in data update is acceptable.

7.2 Cluster Computing

A computer cluster is a group of loosely coupled computers that work together closely so that in many respects it can be viewed as though it were a single computer. Clusters are commonly (but not always) connected through fast local area networks. Clusters are usually deployed to improve speed and/or reliability over that provided by a single computer, while typically being much more cost-effective than single computers of comparable speed or reliability [57].

A cluster is a type of parallel or distributed system that:

1. consists of a collection of interconnected whole computers, and
2. is used as a single, unified computing resource [58].

It may not be immediately clear that these definitions effectively separate clusters from other forms of aggregate computing. Cluster computing differs from parallel computing and SMPs (Symmetric Multi-Processor) computing in three principle areas [58]:

Scaling: To scale SMPs, you must do more than simply add processors. The entire system needs to be upgraded too. However, with clusters, you can simply add more computers as needed as each unit comes with its own complete system (memory, IO, etc.)

Availability: If any part of the SMP breaks, the entire SMP fails, no matter how many CPUs there are. However, clusters can (generally) survive the loss of single units as the processing load is simply transferred to the remaining working units.

System management: After you have programmed the processors to cooperate in a SMP, system management is generally simpler as you only have to do it once.

7.3 Grid Computing

Grid computing uses the resources of many separate computers connected by a network (usually the internet) to solve large-scale computation problems. Grid computing offers a model for solving massive computational problems by making use of the unused resources (CPU cycles and/or disk storage) of large numbers of disparate, often desktop, computers treated as a virtual cluster embedded in a distributed telecommunications infrastructure. Grid computing's focus on the ability to support computation across administrative domains sets it apart from traditional computer clusters or traditional distributed computing [57].

Grid computing has the design goal of solving problems too big for any single supercomputer, whilst retaining the flexibility to work on multiple smaller problems. Thus grid computing provides a multi-user environment. Its secondary aims are: better exploitation of the available computing power, and catering for the intermittent demands of large computational exercises.

This implies the use of secure authorization techniques to allow remote users to control computing resources.

Grid computing involves sharing heterogeneous resources (based on different platforms, hardware/software architectures, and computer languages), located in different places belonging to different administrative domains over a network using open standards. In short, it involves virtualising computing resources.

7.4 Functions of Middleware

Middleware consists of software agents acting as an intermediary between different application components. It is used most often to support complex, distributed applications. In a distributed computing system, middleware is defined as the software layer that lies between the operating system and the applications on each site of the system[15]. Applications use intermediate software that resides on top of the operating systems and communication protocols to perform the following functions:

1. Hiding distribution: the fact that an application is usually made up of many interconnected parts running in distributed locations.
2. Hiding the heterogeneity of the various hardware components, operating systems and communication protocols.
3. Providing uniform, standard, high-level interfaces to the application developers and integrators, so that applications can be easily composed, reused, ported and made to interoperate.

[15]Taken from ObjectWeb - http://middleware.objectweb.org.

4. Supplying a set of common services to perform various general purpose functions, in order to avoid duplicating efforts and to facilitate collaboration between applications.

These intermediate layers have come to be known under the generic name of middleware. The role of middleware is to make application development easier, by providing common programming abstractions, by masking the heterogeneity and the distribution of the underlying hardware and operating systems, and by hiding low-level programming details.

8 Data Provenance

The widespread nature of the Internet and the ease with which files and data can be copied and transformed has made it increasingly difficult to determine the origins of a piece of data. The term *data provenance* refers to the process of tracing and recording the origins of data and its movement between databases. With many kinds of data, provenance is not important. However, for scientists focussed on the *accuracy and timeliness* of the data, provenance is a big issue [59].

Provenance allows us to take a quantity of data and examine its *lineage*. Lineage shows each of the steps involved in sourcing, moving and processing the data [66]. In order to provide provenance, all datasets and their transformations must be recorded.

Frew and Bose [60] propose the following requirements for provenance collection:

1. A standard lineage representation is required so lineage can be communicated reliably between systems (currently there is no standard lineage format[16].
2. Automated lineage recording is essential since humans are unlikely to record all the necessary information manually.
3. Unobtrusive information collecting is desirable so that current working practices are not disrupted.

8.1 The Reasons for Provenance

Scientists are often interested in provenance because it allows them to view data in a derived view and make observations about its quality and reliability [61]. Goble [62] presents some notable uses for provenance:

Reliability and quality: Given a derived dataset, we are able to cite its lineage and therefore measure its credibility. This is particularly important for data produced in scientific information systems.

[16] At least at the time of the publication of [60].

Justification and audit: Provenance can be used to give a historical account of when and how data has been produced. In some situations, it will also show why certain derivations have been made.

Re-usability, reproducibility and repeatability: A provenance record not only shows how data has been produced, it provides all the necessary information to reproduce the results. In some cases the distinction between repeatability and reproducibility must be made. In scientific experiments results may be different due to observational error or processing may rely on external and volatile resources.

Change and evolution: Audit trails support the implementation of change management.

Ownership, security, credit and copyright: Provenance provides a trusted source from which we can procure who the information belongs to and precisely when and how it was created.

Zhao et al. [69] describes three further purposes for provenance[17] from the viewpoint of the scientist:

1. *Debugging.* Experiments may not produce the desired results. The scientist requires a log of events recording what services were accessed and with which data.
2. *Validity Checking.* If the scientist is presented with a novel result, he/she may wish to perform expensive laboratory-based experiments based on these results. Although sure that the workflow design is valid, she may still want to check how this data has been derived to ensure it is worthy of further investigation.
3. *Updating.* If a service or dataset used in the production of a result has changed, the scientist will need to know what implications that change has on those results.

8.2 Data Provenance for the Life Sciences

Among the sciences, the field of molecular biology has generated a wealth of biological data and is arguably one of the most sophisticated consumers of modern database technology [63]. The field of molecular biology supports many hundreds of public databases, but only a handful of these can be considered to contain "source" data in that they receive experimental data. Many of the databases actually reference themselves. This sounds contradictory until you take into account that much of the value associated to a data source comes from the curation and annotation (conducted by experts) of the data.

Most implementors and curators of scientific databases would like to record provenance, but current database technology does not provide much help in this process as databases are typically rigid structures and do not allow the kinds of *ad hoc* annotations that are often needed to record provenance [59].

[17] These are not entirely exclusive to those described by Goble [62].

There have been some attempts to formally monitor data provenance. *Query inversion* is a process that attempts to establish the origin of data by inverting the query on the output tuple. Even for this basic operation, the formalisation of the notion of data provenance is a non-trivial problem [59]. [64] presents a data model where the location of any piece of data can be uniquely described as a path.

One of the most significant problems associated with data provenance can be seen with the following example[18]; Suppose A cites a (component of a) document B. Suppose the owner of B wishes to update it, thereby invalidating the citation in A. Whose responsibility is it to maintain the integrity of B? This is a common problem in scientific (in particular, biological) databases. The usual procedure is to release successive versions of a database as separate documents.

8.3 Data Provenance and Workflow Enactment for the Web and Grid

It is not only the biological science domain that is concerned with data provenance. Large-scale, dynamic and open environments such as the Grid and web services build upon existing computing infrastructures to supply dependable and consistent large-scale computational systems [65]. With both scientific experiments and business transactions, the notion of lineage and dataset derivation is of paramount importance since without it, information is potentially worthless.

[65] proposes an infrastructure level support for a provenance recording capability for service-oriented architectures such as the Grid and web services. Interestingly, [65] also proposes a method that uses provenance for determining whether previously computed results are still up to date.

A provenance capability in a Grid or web services environment has two principal functions: to record provenance on dataset transformations executed, e.g. during workflow enactment, and to expose this provenance data in a consistent and logical format via a query interface.

Provenance needs to be stored and [65] envisages two possible solutions for storage.

1. Data provenance is held alongside the data as metadata.
2. Data provenance can be stored in a dedicated repository, made accessible as a Grid or web service.

The first solution requires the holders of any such data to maintain the integrity of the provenance records as transformations take place, and it imposes significant changes to any existing data storage structures. Such a kind of provenance can be useful for a user to remember how a result was derived, and what steps were involved. It is unclear that such provenance can be trusted by any third party, since the data and provenance owner has to be trusted to have recorded provenance properly.

[18] Taken from [59].

Workflow enactment is the automation of a process during which documents, information or tasks are passed from one participant to another for action, according to a set of declarative or procedural rules [65]. In Grid applications, this task is often performed by a "workflow enactment engine", which uses a workflow script, such as WSFL or BPEL4WS, to determine which services to call, the order to execute them in and how to pass datasets between them.

8.4 The ^{my}Grid Project

^{my}Grid, as a pilot e-science project, aims to provide middleware services not only to automate the execution of *in silico* experiments as workflows in a Grid environment, but also to manage and use results from experiments [67]. The ^{my}Grid project is currently being developed using several molecular biological scenarios.

^{my}Grid is being used to automate complex and tedious *in silico* experimentation[19] by wrapping each web-based analysis tool and data resource as a web service. Within this process, often discarded intermediate results are assigned identifiers and published locally, so that they become resources of the personal web of science [68].

If provenance information is to be shared within ^{my}Grid, we need to overcome their heterogeneity and agree a common understanding (or semantics) as to what the contents of each data item and service represents and the relationships we provide between resources [69].

The following figure from Zhao [69] shows the architecture of the provenance providing components in ^{my}Grid and how provenance data are drawn together.

> From Zhao [69]:
>
> First the user starts a workflow using the Taverna workflow workbench[20]. The workbench holds organisation in- formation such as user identity, which is passed to the workflow enactor together with input data and workflow specification. As the workflow is run, the enactor stores data (using the mySQL RDBMS), and metadata (in a Jena RDF repository)[21] corresponding to our four views of provenance. Each resource (including person, organisation, data and service) is assigned an LSID and made available via an LSID authority implemented using an open source framework[22]. Client applications can then visualize both metadata and data using the LSID protocol.

[19] Often, the mundane nature of the task means not all possible avenues are explored, because the scientist either misses possible routes or discards apparently uninteresting results.

[20] Taverna and FreeFluo are both open source projects available from http://taverna. sourceforge.net and http://freefluo.sourceforge.net

[21] Jena is an open source semantic web framework for Java including an RDF repository http://jena.sourceforge.net/

[22] A LSID Java server stack available for download at: http://www-124.ibm.com/ developerworks/projects/lsid

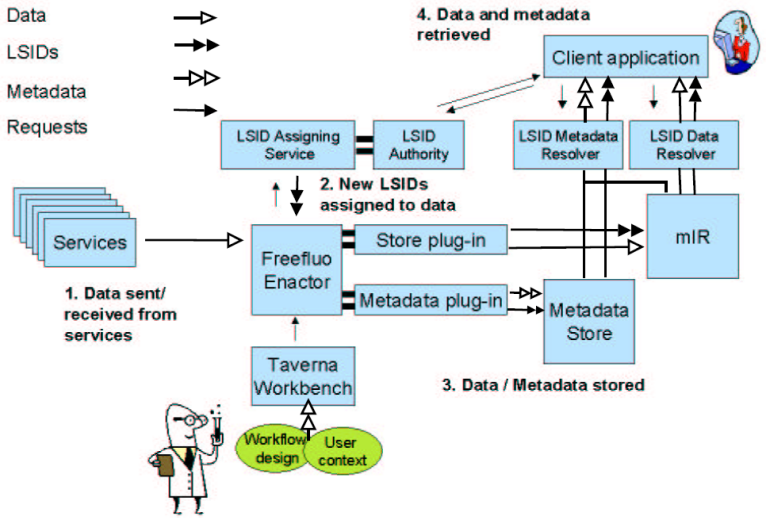

Figure 3: Architecture for Provenance Generation and Visualisation in ^{my}Grid.

8.5 The Taverna e-Science Workbench

This subsection is taken from the ^{my}Grid user guide (bundled with the ^{my}Grid download) and provides a nice description of the Taverna workbench.

The main user interface to myGrid is the Taverna e-science workbench. Taverna provides a language and software tools to enable the design and implementation of workflows. In a bioinformatics context, a workflow is the entire process of collecting relevant data, performing any number of analyses over the data and extracting biological results. Often, in bioinformatics the result of one experiment can form the input values of the next. Designing workflows in Taverna allows services (or bioinformatics analyses) to be scheduled to run in series. If results are not dependent upon one another, services can be designed to run concurrently, allowing for faster, more efficient processing.

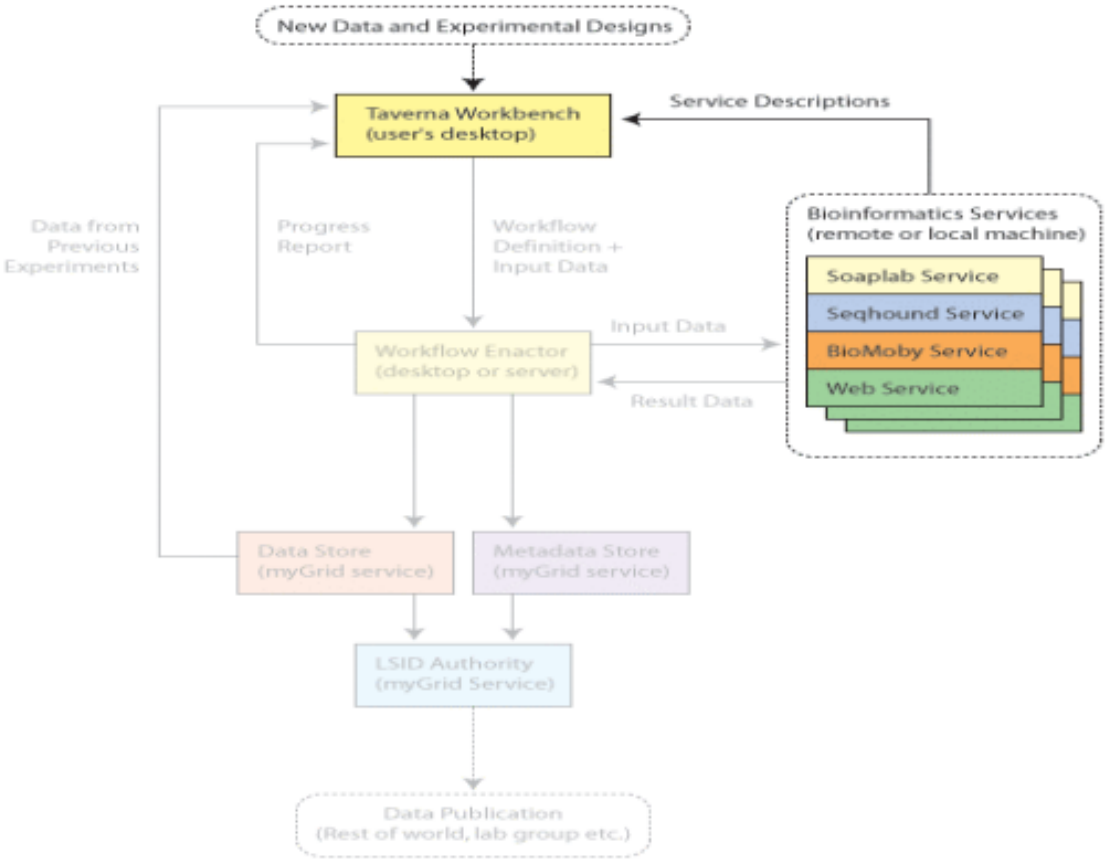

Figure 4: Service selections from remote and local machines

Workflows are designed in Taverna by selecting services and inserting them into a workflow diagram. Services are any bioinformatics applications that are available as web services. More and more applications are being provided as web services, but 'legacy' applications, either command-line tools, or simple web interfaces, can also be transformed into web services using wrapping tools, such as Soaplab and Gowlab respectively. This functionality means that any bioinformatics experiment that can be conducted elsewhere, regardless of the number of different processes or data sources required, can be run as a workflow in Taverna.

References

[1] A. Brazma et al. Minimum information about a microarray experiment (MIAME)-toward standards for microarray data. *Nat Genet, vol. 29, pp. 365-71*, 2001.

[2] A. L. Rector, J. E. Rogers, P. E. Zanstra, and E. Van Der Haring. OpenGALEN: open source medical terminology and tools. *AMIA Annu Symp Proc, pp. 982*, 2003.

[3] J. Futrelle, S. Chen and K. C. Chang. NBDL: a CIS framework for NSDL. *In Proceedings of the 1st ACM/IEEE-CS Joint Conference on Digital Libraries (US)*. JCDL 2001.

[4] Z. Lacroix, Biological data integration: Wrapping data and tools. *Ieee Transactions on Information Technology in Biomedicine, vol. 6, pp. 123-128*, 2002.

[5] S. P. Shah, Y. Huang, T. Xu, M. M. Yuen, J. Ling, and B. F. Ouellette, Atlas - a data warehouse for integrative bioinformatics. *BMC Bioinformatics, vol. 6, pp. 34*, 2005.

[6] J. Koh et al. BioWare: A framework for bioinformatics data retrieval, annotation and publishing. *SIGIR 2004 Workshop*

[7] C. Schönbach, P. Kowalski-Saunders, and V. Brusic. Data warehousing in molecular biology. *Brief Bioinform, vol. 1, pp. 190-8*, 2000.

[8] A. Kasprzyk, D. Keefe, D. Smedley, D. London, W. Spooner, C. Melsopp, M. Hammond, P. Rocca-Serra, T. Cox, and E. Birney. EnsMart: A generic system for fast and flexible access to biological data. *Genome Research, vol. 14, pp. 160-169*, 2004.

[9] D. Gilbert. Shopping in the genome market with EnsMart. *Briefings in Bioinformatics, software review submission*, June 2003

[10] S. M. J. Searle, J. Gilbert, V. Iyer, and M. Clamp. The Otter annotation system. *Genome Research, vol. 14, pp. 963-970*, 2004.

[11] J. A. Cuff, G. M. P. Coates, T. J. R. Cutts, and M. Rae. The Ensembl computing architecture. *Genome Research, vol. 14, pp. 971-975*, 2004.

[12] T. Hubbard. Ensembl 2005. *Nucleic Acids Research, vol. 33, pp. D447-D453* 2005.

[13] R. D. Dowell, R. M. Jokerst, A. Day, S. R. Eddy, and L. Stein. The distributed annotation system. *BMC Bioinformatics, vol. 2, pp. 7*, 2001.

[14] P. I. Olason. Integrating protein annotation resources through the Distributed Annotation System. *Nucleic Acids Res, vol. 33, pp. W468-70*, 2005.

[15] S. Yang, S. S. Bhowmick, and S. Madria. Bio2X: a rule-based approach for semi-automatic transformation of semi-structured biological data to XML. *Data and Knowledge Engineering, vol. 52, pp. 249-271*, 2005.

[16] O. G. Troyanskaya, K. Dolinski, A. B. Owen, R. B. Altman, and D. Botstein. A Bayesian framework for combining heterogeneous data sources for gene function prediction (in Saccharomyces cerevisiae). *Proceedings of the National Academy of Sciences of the United States of America, vol. 100, pp. 8348-8353,* 2003.

[17] K. G. Herbert, N. H. Gehani, W. H. Piel, J. T. L. Wang, and C. H. Wu. BIO-AJAX: An extensible framework for biological data cleaning. *Sigmod Record, vol. 33, pp. 51-57*, 2004.

[18] A.J. Gilliland-Swetland. Setting the Stage. *Introduction to Metadata, Pathways to Digital Information - http://www.getty.edu/research/institute/standards/intrometadata/*, 2000

[19] Craig A. N. Soules, Garth R. Goodson, John D. Strunk, and Greg Ganger. Metadata efficiency in versioning file systems. *Conference on File and Storage Technologies (San Francisco, CA).* 31 March–02 April 2003.

[20] D. Grune, B. Berliner, and J. Polk. Concurrent Versioning System, *http://www.cvshome.org.*

[21] K. R. Dittrich, D. Tombros and A. Geppert. Databases in Software Engineering: A Roadmap. *The Future of Software Engineering, Anthony Finkelstein (Ed.), ACM Press* 2000

[22] B. R. Barkstrom. Data product configuration management and versioning in large-scale production of satellite scientific data. *Software Configuration Management, vol. 2649, pp. 118-133*, 2003.

[23] W.F. Tichy. Software development control based on system structure description. *PhD Thesis.* Carnegie-Mellon University, Pittsburgh, PA, January 1980.

[24] John D. Strunk, Garth R. Goodson, Adam G. Pennington, Craig A. N. Soules, Gregory R. Ganger. Intrusion Detection, Diagnosis, and Recovery with Self-Securing Storage. *Technical report CMU-CS-02-140.* Carnegie-Mellon University, Pittsburgh, PA, 2002

[25] Kiran-Kumar Muniswamy-Reddy, Charles P. Wright, Andrew Himmer, and Erez Zadok. A versatile and User-Oriented Versioning File System.

Proceedings of the Third USENIX Conference on File and Storage Technologies FAST(2004)

[26] D. G. Korn and E. Krell. The 3-D File System. *In Proceedings of the USENIX Summer Conference, pages 147156.* Summer 1989.

[27] D. K. Gifford, R. M. Needham, and M. D. Schroeder. The Cedar File System. *Communications of the ACM, 31(3):288298*, 1988.

[28] BM Rational. Rational clearcase. *www.rational.com/products/clearcase/index.jsp.*

[29] D. Hitz, J. Lau, and M. Malcolm. File System Design for an NFS File Server Appliance. *In Proceedings of the USENIX Winter Technical Conference, pages 235245*, January 1994.

[30] Zachary N. J. Peterson and Randal C. Burns. Ext3cow: The design, Implementation, and Analysis of Metadata for a Time-Shifting File System. *Technical Report HSSL-2003-03*, Computer Science Department, The Johns Hopkins University, 2003. http://hssl.cs.jhu.edu/papers/peterson-ext3cow03.pdf.

[31] Digital Equipment Corporation. *TOPS-20 Users Guide (Version 4)*, January 1980.

[32] K. McCoy. VMS File System Internals. *Digital Press*, 1990.

[33] D. S. Santry, M. J. Feeley, N. C. Hutchinson, A. C. Veitch, R.W. Carton, and J. Ofir. Deciding When to Forget in the Elephant File System. *In Proceedings of the 17th ACM Symposium on Operating Systems Principles, pages 110-123*, December 1999.

[34] J. D. Strunk, G. R. Goodson, M. L. Scheinholtz, C. A. N. Soules, and G. R. Ganger. Self-Securing Storage: Protecting Data in Compromised Systems. *In Proceedings of the 4th Symposium on Operating Systems Design and Implementation, pages 165-180*, October 2000.

[35] S. Khaddaj, A. Adamu and M. Morad. Object Versioning and Information Management. *Information and Software Technology, Volume 46, Issue 7.* June 2004.

[36] Dadam, V. Lum and H.D. Werner. Integrating of time versions into relational database systems. *Proceeding of the Conference on Very Large Database* (1984) pp. 509522 .

[37] F. Grandi, F. Mandreoli, A formal model for temporal schema versioning in object-oriented databases. *A Timecenter Technical Report TR-68*, 2002.

[38] M. Golfarelli, J. Lechtenborger, S. Rizzi, and G. Vossen. Schema versioning in data warehouses. *Conceptual Modeling for Advanced Application Domains, Proceedings, vol. 3289, pp. 415-428,* 2004.

[39] M. D. Flouris and A. Bilas. Clotho: Transparent Data Versioning at the Block I/O Level. *citeseer.ist.psu.edu/flouris04clotho.html*, 2004.

[40] N. C. Hutchinson, S. Manley, M. Federwisch, G. Harris,D. Hitz, S. Kleiman, and S. OMalley. Logical vs. Physical File System Backup. *In Proc. of the 3rd USENIX Symposium on Operating Systems Design and Impl. (OSDI99)*, Feb. 1999.

[41] J. Kovse and C. Gebauer. VS-Gen: A case study of a product line for versioning systems. *Generative Programming and Component Engineering 2004, Proceedings, vol. 3286, pp. 396-415*, 2004.

[42] R. H. Katz. Toward a Unified Framework for Version Modeling in Engineering Databases. *Computing Surveys, vol. 22, pp. 375-408*, 1990.

[43] J. S. Goonetillake, T. W. Carnduff, and W. A. Gray. An integrity constraint management framework in engineering design. *Computers in Industry, vol. 48, pp. 29-44*, 2002.

[44] T. R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *International Journal of Human-Computer Studies, vol. 43, pp. 907-928*, 1995.

[45] OBO: Open Biomedical Ontologies. *http://obo.sourceforge.net*

[46] B. Smith et al. Relations in biomedical Ontologies. *Genome Biology* 2005

[47] W. Ceusters, I. Desimpel, B. Smith, S. Schulz. Using Cross-Lingual Information to Cope with Underspecification in Formal Ontologies. *Proceedings of Medical Informatics Europe* 2003.

[48] J. Verschelde, M. Casella Dos Santos, T. Deray, B. Smith and W. Ceusters. Ontology-assisted database integration to support natural language processing and biomedical data-mining. *Journal of Integrative Bioinformatics*

[49] W. Ceusters, B. Smith, J. M. Fielding. LinkSuite: formally robust ontology-based data and information integration. *Database Integration in the Life Sciences* 2004.

[50] M. Klein and D. Fensel. Ontology versioning for the Semantic Web. *In Proceedings of the International Semantic Web Working Symposium (SWWS), Stanford University, California, USA*, July 30 – Aug. 1, 2001.

[51] N. F. Noy, S. Kunnatur, M. Klein, and M. A. Musen. Tracking changes during ontology evolution. *Semantic Web - Iswc 2004, Proceedings, vol. 3298, pp. 259-273*, 2004.

[52] K. H. Cheung, K. Y. Yip, A. Smith, R. deKnikker, A. Masiar, and M. Gerstein. YeastHub: a semantic web use case for integrating data in the life sciences domain. *Bioinformatics, vol. 21, pp. I85-I96*, 2005.

[53] S. Decker, S. Melnik, F. Van Harmelen, D. Fensel, M. Klein, J. Broekstra, M. Erdmann, and I. Horrocks. The semantic Web: The roles of XML and RDF. *Ieee Internet Computing, vol. 4, pp. 63-74*, 2000.

[54] S. Martin, M. Senger and M. Niemi. Life Sciences Identiers second revised submission. *Technical report, I3C* (2003)

[55] J. Cardoso and A. Sheth. Introduction to semantic web services and web process composition. *Semantic Web Services and Web Process Composition, vol. 3387, pp. 1-13*, 2005.

[56] M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara. Importing the semantic web in UDDI. *Web Services, E-Business, and the Semantic Web, vol. 2512, pp. 225-236,* 2002.

[57] Wikipedia Encyclopedia - http://en.wikipedia.org

[58] G. F. Pfister. In search of clusters (2nd ed.). *Prentice-Hall, Inc.* 1998

[59] P. Buneman, S. Khanna, and W. C. Tan. Data provenance: Some basic issues. *Fst Tcs 2000: Foundations of Software Technology and Theoretical Computer Science, Proceedings, vol. 1974, pp. 87-93*, 2000.

[60] J. Frew and R. Bose. Lineage issues for scientific data and information. *Data provenance/derivation workshop.* October 2002.

[61] P. Buneman, S. Khanna, and Wang-Chiew Tan. Computing provenance and annotations for views. *Data provenance/derivation workshop.* October 2002.

[62] C. Goble. Position statement: Musings on provenance, workflow and (semantic web) annotations for bioinformatics. *Data provenance/derivation workshop.* October 2002.

[63] S. Davidson, C. Overton and P. Buneman. Challenges in Integrating Biological Data Sources. *Journal of Computational Biology, 2(4):557-572*, Winter 1995.

[64] P. Buneman, A. Deutsch, and W. Tan. A Deterministic Model for Semistructured Data. *In Proc. of the Workshop On Query Processing for Semistructured Data and Non-standard Data Formats, pages 14-19*, 1999.

[65] M. Szomszor and L. Moreau. Recording and reasoning over data provenance in web and grid services. *On the Move to Meaningful Internet Systems 2003: Coopis, Doa, and Odbase, vol. 2888, pp. 603-620,* 2003.

[66] D. Pearson. Data requirements for the grid. *Scoping Study Report.* February 2002. Status Draft.

[67] R. Stevens, A. Robinson and C. Goble. myGrid: Personalised Bioinformatics on the Information Grid. *International Conference on Intelligent Systems in Molecular Biology.* (2003)

[68] J. Hendler. Communication: Enhanced Science and the Semantic Web. *Science 299* (2003) 520-521

[69] J. Zhao, C. Wroe, C. Goble, R. Stevens, D. Quan, and M. Greenwood. Using semantic web technologies for representing e-Science provenance. *Semantic Web - Iswc 2004, Proceedings, vol. 3298, pp. 92-106*, 2004.