Universidade de Aveiro
2009

Departamento de Electrónica,
Telecomunicações e Informática

João Eduardo de
Sousa e Almeida
Pereira

GeNS: uma Plataforma de Integração de Dados Biológicos

GeNS: the Genomic Name Server

Universidade de Aveiro
2009

Departamento de Electrónica,
Telecomunicações e Informática

João Eduardo de
Sousa e Almeida
Pereira

GeNS: the Genomic Name Server

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Professor Doutor José Luís Guimarães Oliveira, Professor Associado do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

Para a minha família e para a Miguel, por algo que não posso exprimir em palavras.

o júri

presidente

Doutor Joaquim Arnaldo Carvalho Martins
Professor Catedrático da Universidade de Aveiro

Doutor Rui Pedro Sanches de Castro Lopes
Professor Adjunto do Departamento de Informática e Comunicação da Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Bragança

Doutor José Luís Oliveira
Professor Associado da Universidade de Aveiro

agradecimentos

Ao Professor José Luís Oliveira, ao Joel Arrais e ao João Fernandes pelos conselhos, críticas e sugestões durante o desenvolvimento deste trabalho; à minha família e aos meus amigos por todo o incentivo e apoio que me deram ao longo do ano.

Gostaria ainda de agradecer ao Luís Santos pelo seu contributo na análise DiseaseCard vs GeNS (e inúmeros esclarecimentos na área de Biologia) e ao resto do grupo de Bioinformática pelo seu companheirismo.

Por fim, a todos aqueles cuja memória me possa ter falhado.

palavras-chave

Integração de dados, base de dados biologicas, data warehousing, extracção de dados, olap, soap, service, xml, xpath, xsdl, web services

resumo

Os desenvolvimentos científicos vindo do campo da biologia molecular dependem em grande parte da capacidade de análise de resultados laboratoriais por parte de aplicações informáticas. Uma análise completa de uma experiência requer, tipicamente, o estudo simultâneo dos resultados obtidos a par com dados disponíveis em várias bases de dados públicas. Fornecer uma visão unificada deste tipo de dados tem sido um problema fundamental na investigação ao nível de bases de dados desde o aparecimento da Bioinformática.

Esta dissertação apresenta o GeNS, um data warehouse híbrido com uma abordagem simples e inovadora que pretende resolver diversos problemas de integração de dados biológicos.

keywords

Data integration, biological databases, data warehousing, data extraction, olap, soap, service, xml, xpath, xsdl, web services

abstract

The scientific achievements coming from molecular biology depend greatly on the capability of computational applications to analyze the laboratorial results. A comprehensive analysis of an experiment requires, typically, the simultaneous study of the obtained results with data that is available from distinct public databases. Being able to provide a unified view of this data has been a fundamental problem in database research since the dawn of Bioinformatics.

This dissertation introduces GeNS, a hybrid data warehouse that presents a simple, yet innovative approach to address several biological data integration issues.

Table of Contents

Table of Figures

Index of Tables

Abbreviations

DB	*Database*
DBMS	*Database Management System*
DTA	*Database Tuning Advisor*
DNA	*Deoxyribonucleic Acid*
EBI	*European Bioinformatics Institute*
EMBL	*European Molecular Biology Laboratory*
GUI	*Graphical User Interface*
HTML	*HyperText Markup Language*
HTPP	*HyperText Transfer Protocol*
IIS	*Internet Information Services*
I/O	*Input/Output*
JSON	*JavaScript Object Notation*
KEGG	*Kyoto Encyclopaedia of Genes and Genomes*
NCBI	*National Center for Biotechnology Information*
OLAP	*Online Analytical Processing*
OLTP	*Online Transaction Processing*
PDB	*Protein Database*
RDA	*Remote Data Access*
REST	*Representational State Transfer*
SKB	*Source Knowledge Base [BioMediator]*
SOAP	*Simple Object Access Protocol*
SQL	*Structured Query Language*
SWOT	*Strengths, Weaknesses, Opportunities, Threats*
URL	*Uniform Resource Locator*
WS	*Web Service*
WSDL	*Web Services Description Language*
WWW	*World Wide Web*
XML	*Extensible Markup Language*

Chapter 1 - Introduction

1.1. Bioinformatics: an historical perspective

Over the last thirty years, several advances in molecular biology and in genomic technologies, such as the Sanger sequencing method [1] or the polymerase chain reaction [2] technique, for example, have been pushing the boundaries of life sciences. The need to store and catalogue an ever-increasing amount of biological information was a daunting task for which the traditional integration methods were simply too cumbersome to be of practical use. This issue, along with the propagation of affordable personal computers and graphical workstations, led to the creation of computerized databases and specialized tools for viewing, maintaining and analyzing data. A new and exciting field of research and development had emerged: Bioinformatics, a term popularized by Paulie Hogeweg [3] in 1978.

These advances paved the way for the Human Genome Program [4] (HGP), an international scientific research project that aimed to map all the genes in the human genome. This historical effort sparked the Genomic Revolution [5] around 1999, which translated into an enormous increase of the number of mapped genes (Figure 1.1, taken from [5]); by 2003, the genome map was virtually complete.

This discovery is one of the greatest achievements in Bioinformatics and the human genome map has become the cornerstone of many research projects, particularly in molecular biology; it is and will be a vital component in the development of new medical technologies, for example. The HGP effectively streamlined the field of Bioinformatics, which kept evolving at an astonishing rate and, eventually, established itself as one of the most promising and exciting areas of research and development.

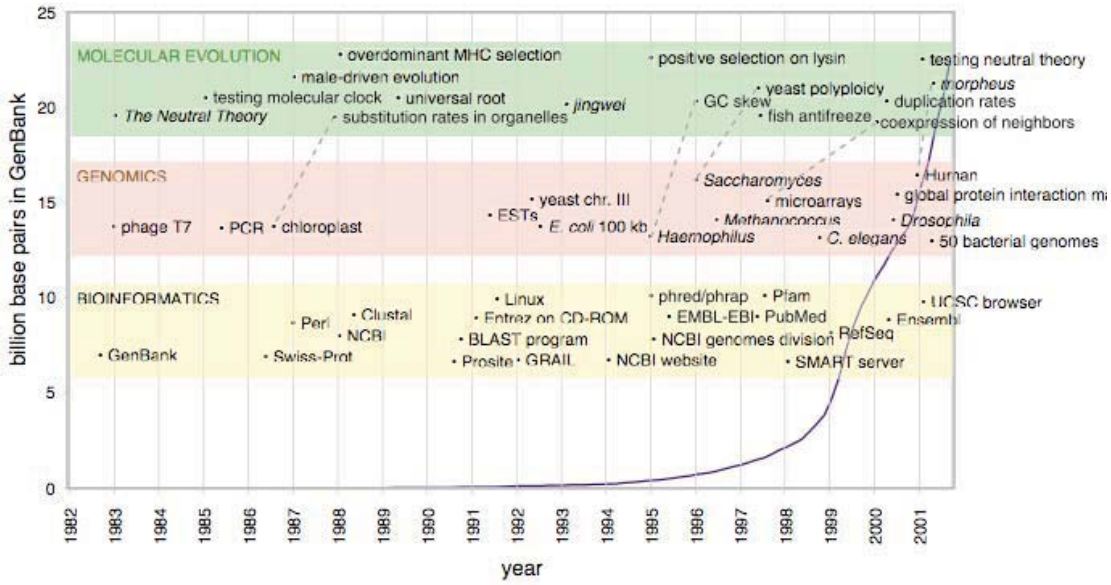

Figure 1.1 – A timeline showing the Genomic Revolution's effects on GenBank [5]

1.2. Overview

According to the latest release of the Nucleic Acids Research there are about 1170 databases in the field of molecular biology [6]. Each database corresponds to the output of a specific study or community and represents a huge investment whose potential has not been fully explored. For a scientist working in the area of molecular biology, being able to retrieve, analyze and combine data from multiple sources is of vital importance, as the data about one biological entity may be dispersed over several databases. For instance, for a gene, the nucleotide sequence is stored in GenBank [7], the pathway in KEGG Pathway [8] and the expression data in ArrayExpress [9]. By placing several distinct pieces of information in context, a much broader view is achieved. Hence, in order to fully understand the role of a gene, a unified view of the data is required and this task may require a considerable amount of time, as it is manually performed.

The integration of heterogeneous data sources has been a fundamental problem in database research since the dawn of Bioinformatics [10-15]. The goal is to achieve better methods to combine data residing at different sources, under different schemas and with different formats in order to provide the user with a unified view of the data. Although simple in principle, this is a very challenging task where both the academic and the commercial communities have been working and proposing several solutions that span a wide range of fields. Notwithstanding, no

solution is perfect and the limitations found globally across solutions (extensively discussed along this document) reflect the difficulty to obtain a simple but comprehensive schema capable of accommodating the heterogeneity of the biological domain while maintaining an acceptable level of performance.

1.3. Objectives

The objective of this work is to develop a biological data integration platform called GeNS (Genomic Name Server) that provides centralized access to heterogeneous data distributed across public databases, as well as the possibility to act as a name server by converting identifiers from a given gene.

This task rises up a set of technical challenges such as what is the best integration strategy, how to solve nomenclature clashes, how to solve database-overlapping data and how to deal with huge data sets. In addition, the system should have an easy to understand (and maintain) relational database schema, and flexible enough to allow the integration of large set of heterogeneous biological data. GeNS' performance is another issue that must be addressed if the system is to be of any practical use.

Furthermore, access layers should be created to allow transparent data integration from external sources. Methods to retrieve, modify and integrate data from external sources (e.g. tabular file parsers, Web Services retrievers and SQL scripts) must be implemented if the system is to provide a unified view of biological data.

Web Services methods are also quite important, as they provide easy and instant access to the data thus skipping the need to have a local copy of the database in order to use the system (while dodging considerable disk space restrictions in the process). Moreover, the Web Services also free users from operating system restrictions.

1.4. Structure

This thesis is divided in five chapters: the second chapter provides a description of the state-of-the-art regarding biological data integration issues and strategies, along with detailed descriptions of third-party data integration tools.

Chapter three, the longest chapter in this thesis, describes past work, GeNS' requirements and the system's architecture. In addition, database enhancements, data integration techniques and data accessibility are also extensively described in this chapter, along with the problems detected while developing the system.

Chapter four describes validation tests regarding GeNS' manual and programmatic utilization.

Finally, chapter five provides a global insight on the accomplished work and points out new possibilities for GeNS regarding future work.

Chapter 2 – Integrating biological data

Conceptual and technical issues hinder heterogeneous biological data integration. These difficulties will be summarized in the following sections, along with a set of integration techniques organized in three main strategies: Links, Mediators and Warehouses. Finally, several preeminent third-party biological data integration tools will be analysed.

2.1. Conceptual Issues

Biological data sets possess certain characteristics prone to raise conceptual problems in the development of a centralized data integration solution.

To begin with, their very nature makes data modelling a hard task. Data modelling is the first step towards the construction of a data integration system. In this process, several data models that accurately define the problem and represent the data (along with their relationships) are created. These models are blueprints that specify which and how elements of the data will be stored inside a database. This process is of crucial importance in the development of any data integration system, especially when dealing with heterogeneous biological data; a lot of these system's performance relies on solid, organized, carefully planned - and, usually, hierarchical - data structures.

The inherent complexity of living organisms, along with the complexity of the biological systems themselves, often translates into a large number of concepts inserted into the system, along with the multiple relationships between themselves that must also be stored [16].

Figure 2.1, taken from [17], is a good example; here, the extent of the pathways associated with the citrate cycle is shown. If mapping all of these relationships for this cycle alone is not an easy task by itself, correctly mapping the myriad of pathways, related with a given organism, for all of its cycles is an immensely more complex task. Due to the heavy number of relationships between concepts, this kind of databases tends to grow at an impressive rate, easily reaching hundreds of millions of stored objects and gobbling up several hundred gigabytes of disk space. Moreover, new types of data continue to appear on a regular basis.

As a consequence, and quoting Frédéric Achard et all [18]: "*Not only must these new types of data be modelled properly but they also modify our perception of the old types of data*".

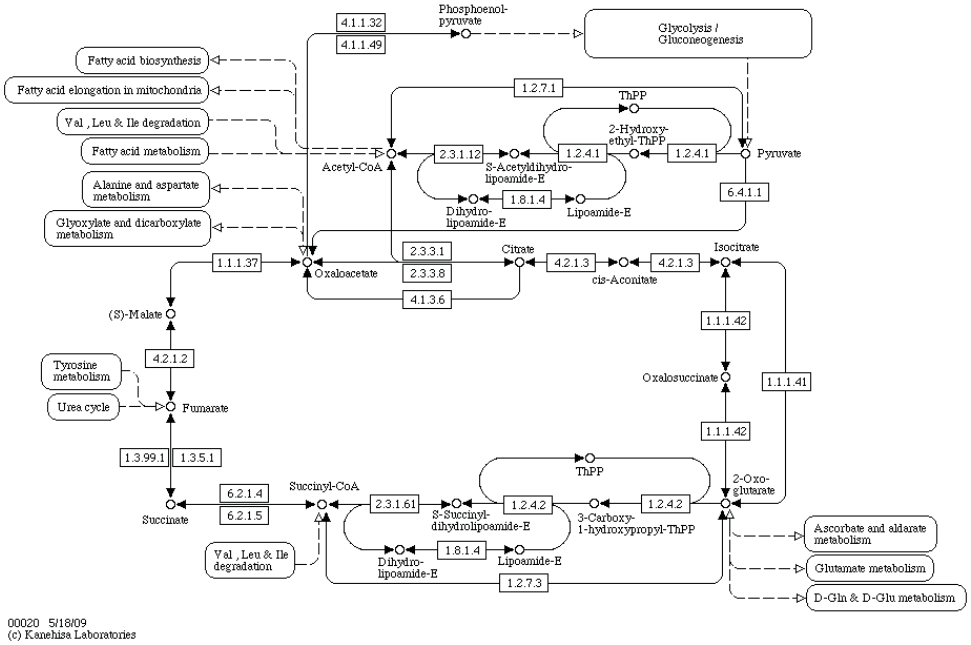

Figure 2.1 –A pathway map for the citrate cycle [17]

This means that the system must be flexible enough to allow the insertion of new concepts and provide means to update and/or delete all the objects related to old types of data being replaced by newer ones. This is especially important, as this kind of data is updated quite often (e.g., Uniprot [19, 20] spawns a new release set approximately every month). Being able to automatically collect and import new sets of data is also crucial to any data integration system because manually performed tasks can become rather costly, especially when dealing with this kind of complex, large sets of data.

Additionally, post-integration data processing methods – generating new, computationally derived data that must also be integrated – are a need for many scientists in this field.

The way in which all these objects and relationships are mapped is, beyond a shadow of a doubt, vital for the system's success. One of two main strategies are usually followed: hierarchical [21] or graph [22] models. The former, as its name indicates, organizes the concepts hierarchically; while easier to comprehend, hierarchical models allow property inheritance from the parent concepts and tend to translate themselves to well-defined, expandable systems.

Graph models, on the other hand, allow for a much greater flexibility, as concepts no longer need to be as strictly categorized and ordered as in hierarchical models. This is quite important, as heterogeneous biological data does not easily fit into a hierarchical schema.

The advantages/disadvantages of these two systems are, in fact, a dichotomy: while the former is easy to understand and expand (but may encounter fundamental design problems), the latter's nature makes it the ideal choice for modelling complex data structures and their relationships (at the cost of the system's intelligibility and expandability).

Another possible issue that may arise during this kind of data integration is the organism's taxonomical classification; organizing biological entities according to the organism they belong to is a very reasonable and common approach that allows for a structured view of the data. Each living organism has a taxonomical classification – except for some very specific situations, e.g., a recently discovered organism or a suspected new species – that groups organisms in a hierarchical structure, according to their common ancestors, biological, biochemical and physical characteristics.

This taxonomical classification system, called the Linnaean taxonomic system, is based on the work of the Swedish biologist Carl Linnaeus and groups organisms into seven main categories[1]: Kingdom, Phylum, Class, Order, Family, Genus and Species.

Figure 2.2 shows Linnaeus' table of the Animal Kingdom from the first edition of Systema Naturae [23]. Two hundred and seventy four years later, the taxonomic tree has grown so large that it can no longer be assembled in a single book: NCBI taxonomy statistics shows that their entire taxonomy tree had over 325.000 nodes in May, 2009 (a number that has been growing steadily over the past few years).

As a comparative example, Figure 2.3 presents a segment of a taxonomical tree-view regarding three chosen species, highlighted in bold: humans (*Homo sapiens*), fox squirrels (*Sciurus niger*) and sprot prawns (*Pandalus platyceros*). The fourth field in bold – *Sciurus niger rutiventer* – is actually a subspecies of the fox squirrel. A subspecies is a taxonomic subdivision of a species and, as such, cannot exist without one; it represents a group of organisms separated from an original population of a given species.

It is a product of the continual process of evolution and, given enough time, a subspecies will eventually become a species (a descendent of the original species it was once a part of).

[1] Although several intermediate super- and subdivisions exist inside these main categories, they are not listed for simplification purposes.

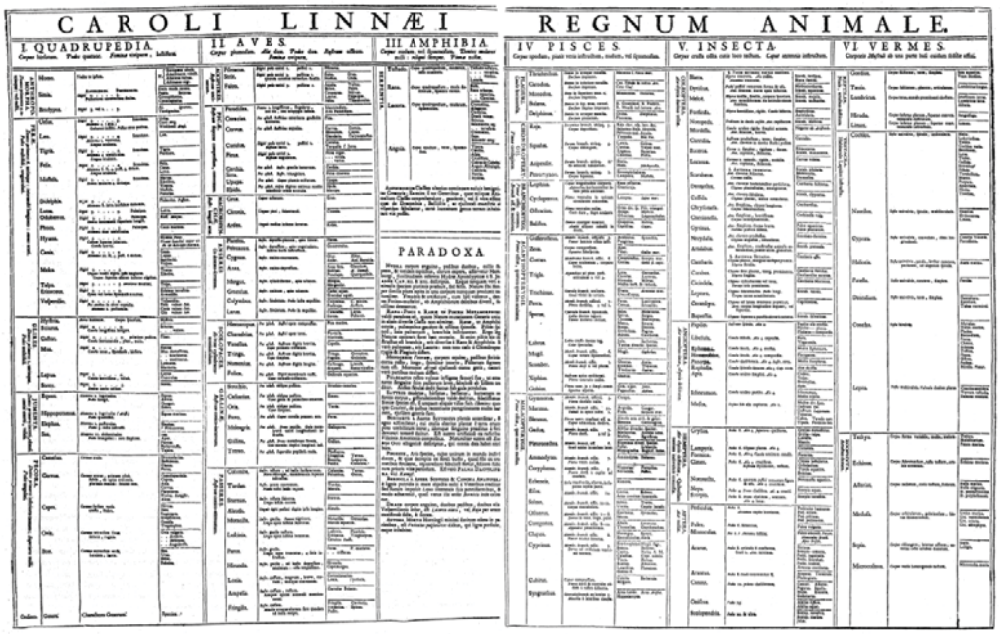

Figure 2.2 - Carl Linnaeus' taxonomic tree for the Animal Kingdom

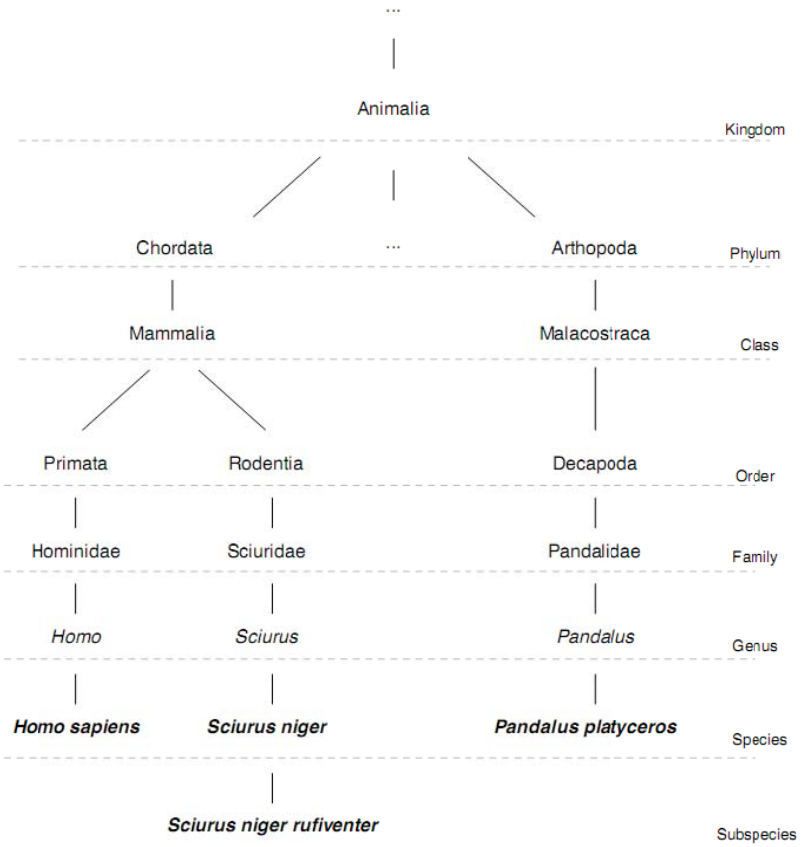

Figure 2.3 - A taxonomical comparison between humans, fox squirrels and sprot prawns

Although NCBI's excellent taxonomy tree can be freely downloaded from their website, any heterogeneous data integration system must have a database capable of incorporating not just the current taxonomic tree data, but also future versions of it. On a more practical level, this would include introducing new species into the tree and deleting old ones, always keeping the coherence of the data. Hence, if a data integration solution is to incorporate this kind of information, content scalability is, once more, the keyword.

Adding it all up, the peculiarities of heterogeneous biological data sets make data modelling a large-scale organizational task [24], always balancing flexibility and content scalability [25] with performance and simplicity, while attempting to minimize data redundancy and overly complicated database schemas. Up to this day, this task remains a challenge in bioinformatics [14].

2.2. Technical Issues

From a technical point of view, several matters present themselves when developing this kind of systems. To begin with, the biological data sets, spread out across a large number of sources, must be retrieved [6].

The most common ways to access biological data are flat file download via FTP servers, consuming Web Service's methods and remote database access by SQL. Although each way has its own advantages, disadvantages and typical use cases, each source decides which methods to implement and which to ignore, according to its data availability polices. Regarding the first (direct download), UniProt and KEGG, for example, publish their data as compressed flat files on FTP servers. Flat files are plain text files that contain one record per line (usually, from a single table only). This method provides bioinformaticians and users alike easy, fast access to a very large set of data. Nonetheless, if the amount of data to retrieve were small (or even moderate), then this method would not be appropriate because users would still have to wait for the retrieval and parsing of irrelevant data. The flat files format also depends of the source; it can be either in XML format (e.g., Uniprot), tabular - separating the data fields by tabs – format (e.g., KEGG), or even in a very friendly human-readable format such as the one used by Gene Ontology [22] (which, unfortunately, is particularly bad for parsers due to its unstructured form).

Another way to access this data is by means of Web Services. Web Services empower mediator-based data integration systems by entirely skipping the need to locally store information.

KEGG, Biomart [26] and ArrayExpress [27], for example, developed a Web Services API that allows users to query and retrieve information either programmatically or via web browser. This method allows quick access to small or moderate amounts of data. On the other hand, it is inappropriate for larger amounts mainly due to performance-related motives: it is too slow, especially when compared to the transfer rates that can be achieved by direct downloads. Still, Web Services are reasonable alternative due to their "popularity" amongst sources, especially in the absence of better retrieval methods.

Likewise, accessing remote databases via SQL Access [28] is another valid alternative. SQL Access is an implementation of the RDA standard protocol that allows access to resources on remote database servers, as if the database was running locally. Similarly to Web Services, SQL Access enables its users to avoid certain software restrictions (e.g. operating system restrictions) by providing a common interface for data retrieval. In addition, by enabling SQL Access to a remote database its users are able to pose custom, complex queries and to obtain the data directly from the DBMS. PublicHouse - a set of biological databases constructed using the BioWarehouse [29] open-source data integration tool – is an example of such case. As if using the Web Services API's, this method should not be used for large data sets.

Database loaders, responsible for obtaining and parsing data from all external sources of interest, must be implemented for all access methods. They should also be as generic as possible, in order to facilitate integration from new sources and coping with changes in older ones.

Another option is the use of Web Crawlers [30]. Web Crawlers are software programs that browse the World Wide Web methodically, retrieving HTML pages and collecting their content. This content can, subsequently, be indexed either by the Web Crawler itself or by external programs. By repeatedly downloading and processing the content of a large amount of websites, Web Crawlers are very good in data mining operations and, as such, are pivotal tools for some larger systems (e.g. search engines). Due to their *modus operandi,* in which these programs run for long periods of time because of the enormous amount of websites that must be analysed), Web Crawlers can provide an interesting feature for biological data integration systems: continually inserting and/or updating data in real-time without human interaction. If correctly implemented, such a feature could bestow GeNS with an always-updated set of data (with minimum maintenance).

Provided that the system can access and transfer the desired data sets, their integration poses both short and long-term issues. While the former have already been detailed in section 2.1

(conceptual issues regarding how to map some many fundamentally different concepts and the relationships between themselves), the latter refers to the serious physical scalability issues (regarding disk space, memory constraints, etc) [18] imposed by the sheer volume of biological data sets and their continual, extraordinary growth on the long run. This is especially important, as over the past few years high-throughput data sequencing techniques have been producing larger and larger amounts of data [31].

2.3. Data Integration Strategies

Although it is consensual that the use of biological data spread over the web is essential to extract knowledge from local datasets, it isn't always clear what is the best method to access the data [12]. In this section we review a set of integration techniques organized in three main strategies: Link-based, Mediators and Warehouses.

2.3.1. Link-based

Navigational or link-based integration has been the first and the most successful approach to data integration. This approach consists mainly of web-based systems that offer an interface that provides navigation and searching operations across several data sources. The reason for the success of this approach is that it sticks very closely to the nature of the web. In the context of molecular biology the problem is that an increasing number of sources on the web require users to manually browse through several web pages and data sources in order to obtain the desired information. In addition, since each database has its own interface the user has to "learn" how to navigate in every single database.

In order to solve those problems and to simplify the researcher's task, a system that aggregates all the direct links to the existent database is provided, so the user only needs to access a single web site and provide the query string only once, in order to get all the available information about a specific subject. In some implementations no database is used; the identifiers, needed to construct the URL query, are obtained in runtime by parsing the initial query, or the web page.

However, in others implementations of this technique, a local database is used to store the identifiers that have been previously obtained. One of the biggest advantages of those systems is that the data is always updated due to the fact that little or no local data storage is done (according to each previously mentioned variation). As a consequence, the development process is simplified.

However, this approach also has several drawbacks [12] [29]. First, it is very vulnerable to name clashes and ambiguities. If the source database changes the way the URL are constructed the database stays automatically unavailable and these problems are usually only solved with human interaction. Other relevant problem is that some data sources don't provide the access to their databases through the use of URL construction or even others have implementations that make difficult the use of this approach (session management, cookies, etc.). Another drawback of this approach is the impossibility to manipulate the source database in order to extract data from it. So, on the contrary of the warehouse where the smallest data element is, for instance, a gene name, in this case the smallest data element is the web page itself. This approach is used by DiscoveryLink [32] and DiseaseCard [33].

All things considered:

- ✓ Simplest metaphor
- ✓ Data is always updated
- ✓ Low hardware requirements

- – Reliability
- – No data processing can be performed
- – Difficult to implement a version control system

2.3.2. Mediators

The mediator architecture [34] allows the construction of domain-specific, middleware systems. These systems provide a front-end (typically a graphical user interface) to receive queries from the users. These are reformulated in runtime into single or multiple sub queries, which are, in turn, remotely submitted to external heterogeneous data sources. Finally, the result(s) are aggregated into a unified view - the final result that will be returned to the client.

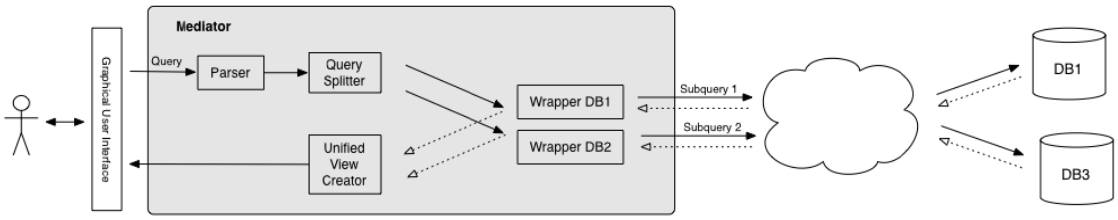

Figure 2.4 - A typical implementation of the mediator architecture

This approach has some interesting perks: since the queries are performed in runtime and the information is directly retrieved from its sources, no data is stored locally and the results are always updated. Moreover, this approach grants a fantastic flexibility to the system, as it can

cope with as many concepts as its sources can store. Simply put, there is no need to organize the data because the system does not have a local database and nothing is locally stored. Instead, the results are already organized in their remote sources. If one were to add a new data source to the system, only the mediator engine (responsible for translating the original query to the sub queries that form the view) would need adjusting.

On the other hand, the lack of performance derived from parsing the original query and retrieving the data in runtime from the sources is a serious drawback, as the system's total delay corresponds to the sum of the initial parsing operation with the delay of each sub query. What's more, the system's reliability is also a considerable disadvantage since all of the results come straight from their sources. If an external data source ceases to be available for any reason at all (e.g., maintenance issues), the mediator based integration system will not be able to retrieve all of the data it is supposed to.

On top of that, the system is also vulnerable to network failures, as it loses contact with the external sources and has no way to present the requested results. Finally, the query translators are difficult to create and update, undermining the system's scalability.

Using the mediator approach is particularly interesting when the sources are considered to be reliable, have moderate to large dimensions and the access is not intensive. Some examples are BioMediator [35] and SEMEDA [36].

Adding it all up:

- ✓ Flexibility (multiple views from the same data)
- ✓ Data is always updated
- ✓ Low hardware requirements

- – Performance
- – Reliability
- – Derived data cannot be stored
- – Scalability
- – Difficult to implement a version control system

2.3.3. Warehouse

A data warehouse is a "subject-oriented, integrated, time-varying, non-volatile collection of data that is used primarily in organizational decision making" [37]. Although at first sight this definition may seem quite similar to that of an ordinary database, data warehouses differ on their purpose: answering abstract, complex questions that cannot be answered by a single

database. Instead, warehouse integration consists in physical integrating data from multiple external sources into a local database and executing all the queries directly on it. In order to use data warehouses, a unified data model that can accommodate heterogeneous information has to be developed, along with software loaders that fetch and transform data (to match the local unified schema) and load them into the warehouse itself.

After this initial setup phase, the warehouse can be used as a single interface to answer any of the questions that the source databases can handle (assuming no information is discarded during the integration process), as well as those that require the interlink of several concepts that are not present in any single database, thus proving a unified view of the data. Data warehouses are high-performance, reliable systems (two key features for biologists and bioinformaticians). In addition, version control is another important factor easily implemented in data warehouses, crucial in large databases.

However, the potential cost associated with the development of this kind of systems should not be taken lightly. As seen in section 2.1, developing a global data schema that encompasses all the desired types of data and their relationships is a hard task. Some usual errors are the development of complex schemas that tend to be difficult to understand, to maintain and to use, or the development of simple and plain schemas that hardly reflect the domain of the problem and be compatible with future database releases. Data replication is also frequent in these databases, as the same entity can exist in multiple sources.

On top of that, new kinds of data types are constantly emerging. This fact, when combined with the occasional changes in an external data source may cause inconsistencies or even errors that (tend to) require human interaction.

Another problem is that once as the data is locally stored, considerable hardware resources - such as disk space and memory - are required. These will grow steeper over time, as the volume of data in UniProt, EMBL and KEGG, for example, keeps increasing over the years. Furthermore, trained personal that will maintain and update the system are also a necessity.

Usually, warehouses are pointed to be best suited for the creation of highly curated datasets focused on a specific and narrow area of research. Successful implementations of warehouses in the domain of molecular biology are Biozon [38] and BioWarehouse [29].

There are two opposing theories regarding how to build a data warehouse: *Online Transaction Processing* (OLTP) [39] and *Online Analytical Processing* (OLAP) systems [40].

OLTP are distributed systems that typically deal with a heavy transactional loads [39] [41]. These systems are used to store critical business tasks and are the primary point of entrance for new data. As such, they store small, always updated sets of data that reflect the current state of the system.

The development of this kind of warehouses follows a thoroughly defined plan, complete with milestones and deadlines. These systems employ tight, rigorous data modeling techniques that fully apply Codd's data normalization rules [42] to break down data to its most simple structures.

As a consequence, the database schema is optimized for consistency, always seeking data integrity and easily creating a large number of tables in the process. New data can be quickly inserted as well as updated, although joining data from distinct tables might prove more difficult than expected due to the resulting complexity of the schema; having to merge data from twenty or more tables is bound to take its toll on the system. As a result, retrieving data from these databases may take longer than their OLAP counterparts [43].

Furthermore, their poor intelligibility is automatically translated into low user-friendliness, making it hard for new users to grasp how the database works. OLTP databases are also frequently content specific, hence incorporating limited amounts of data, and services running on top of them frequently need to access multiple databases at a time in order to merge information on different subjects, thus creating a global panorama.

On the other hand, there's OLAP – *Online Analytical Processing* – based systems. OLAP systems are centralized systems (e.g. data warehouses) that collect data from multiple (OLTP) sources, consolidating and merging data on several aspects [41]. These systems keep track of the status of its sources' data over time, enabling the creation of a timeline keeping track the progress of a given aspect.

Systems running on top of OLAP databases typically do not need to access more than one resource in order to get a unified view of the data. Moreover, their disk space requirements are also much larger due to the need to store old data.

Regarding its database schema and unlike OLTP based systems, this new approach relies not on a detailed, structured plan, but on consecutive iterations that serve to build a thoroughly tested system. Quoting Ronald Gage Allen [43]: *"An effective and robust design can't be planned; it must be iterated"*.

The data is stored either on star or on flake schemas [40] (whether it needs to be ordered hierarchically or not). These schemas favor performance and intelligibility over normalization[2], albeit if the price to pay is a certain level data replication in the database. As a direct result, inserting or updating information in the database requires a greater deal of data transformation, thus taking longer for its completion (compared to OLTP based schemas). However, as merging data from different tables becomes a much easier task, the database is able to retrieve the results faster [43].

While opposing ends of a spectrum several intermediate levels from these two approaches can be drawn, as neither is always the right choice. Instead, the answer lies in the characteristics of the problem itself: these must be identified and carefully weighted in order to determine the best solution towards a unified data integration system.

In sum:

- ✓ Performance
- ✓ Reliability
- ✓ Version control support
- ✓ Post-integration derived data support
- ✓ Ideal for large data sets

- – Steep hardware requirements
- – Maintenance (human interaction is required for updating the data)
- – Hard to design and main database schemas
- – Data loaders must be developed
- – Integration operations suffer from unavoidable long delays in data retrieval and storage; the data must be previously parsed before it can be used

2.4. Third-party Data Integration Solutions

In this section, several preeminent third-party data integration solutions will be described, along with their advantages and disadvantages. These tools will be either link-based, mediator-based or data warehouses, although many successfully manage to accumulate characteristics from mixed architectures.

[2] Data is still normalized, although not as much as in OLTP based schemas

2.4.1. BioWarehouse

BioWarehouse is *"an open source toolkit for constructing bioinformatics database warehouses using the MySQL and Oracle relational database managers"*, developed by SRI International's Artificial Intelligence Centre. A component of the Bio-SPICE project, BioWarehouse's main goal is *"to enable different investigators to create different warehouse instances that combine collections of DBs relevant to their interests"* [29].

BioWarehouse provides its users all the benefits from the data warehousing architecture while attempting to minimize two of its greatest defects: the need to design a stable and correct relational database schema and having to develop software loaders that retrieve, transform and integrate the data.

Currently supporting fourteen distinct databases (including UniProt, KEGG, Gene Ontology and GenBank, among others), the provided schema was designed to integrate several morphologically different types of data (e.g., genes, proteins, nucleic acids, pathways), not only from external data sources but also from locally produced data. However, such flexibility comes at a price: complexity. BioWarehouse's relational database schema has four types of tables: constant tables, object tables, linking tables, and special tables. While the first type – *constant tables* - is used to contain scientific constants (e.g., controlled vocabulary terminology), the second groups all the tables storing entities (i.e. types of data, such as pathways) in the database. The third type is used to store all the relationships between entities and, finally, the fourth type stores data warehousing specific information (encompassing meta-data, loader specifications, cross-references, etc). All in all, nearly two hundred tables exist in this schema, as depicted in Figure 2.5.

Fortunately, the set of tools provided by SRI considerable minimizes this repercussion, as most of the users no longer need to fully understand the schema itself in order to use the system. These tools range from database loaders (written in either C for the MySQL DBMS or Java for the Oracle DBMS) to data mining utilities (written in Perl; several example scripts are also provided to allow the creation of user-customized scripts).

The database loaders parse and transform data found in flat files to the relational database's schema. Along the process, only a minor amount of database-specific information is discarded and, in addition, some of the data suffers negligible changes during its translation to the current schema. These loaders do not tackle the problem of data replication, found when two or more databases contain information about the same element; this task would make data loaders not

only inherently more complex, but also a lot slower hence crippling the performance of the system. In spite of the resulting replication, post-integration redundancy-reduction algorithms can be applied on top of the original database, thus creating a non-redundant view of it.

Figure 2.5, taken from BioWarehouse's documentation section (accessible at http://biowarehouse.ai.sri.com/repos/schema/doc/), illustrates BioWarehouse's database schema. Due to its very large number of tables, making this image comprehensible (visually speaking) would require an entire page and, as such, it is show for illustrative purposes only.

Furthermore, BioWarehouse provides extensive documentation, ranging from quick start guides, to environment settings descriptions, from DBMS settings to developer related information. This is extremely important for a system as complex as this, yet another trait that makes BioWarehouse a very successful heterogeneous data integration tool.

Finally, BioWarehouse can be used either locally or remotely, as several public servers, such as PublicHouse, are freely available for use.

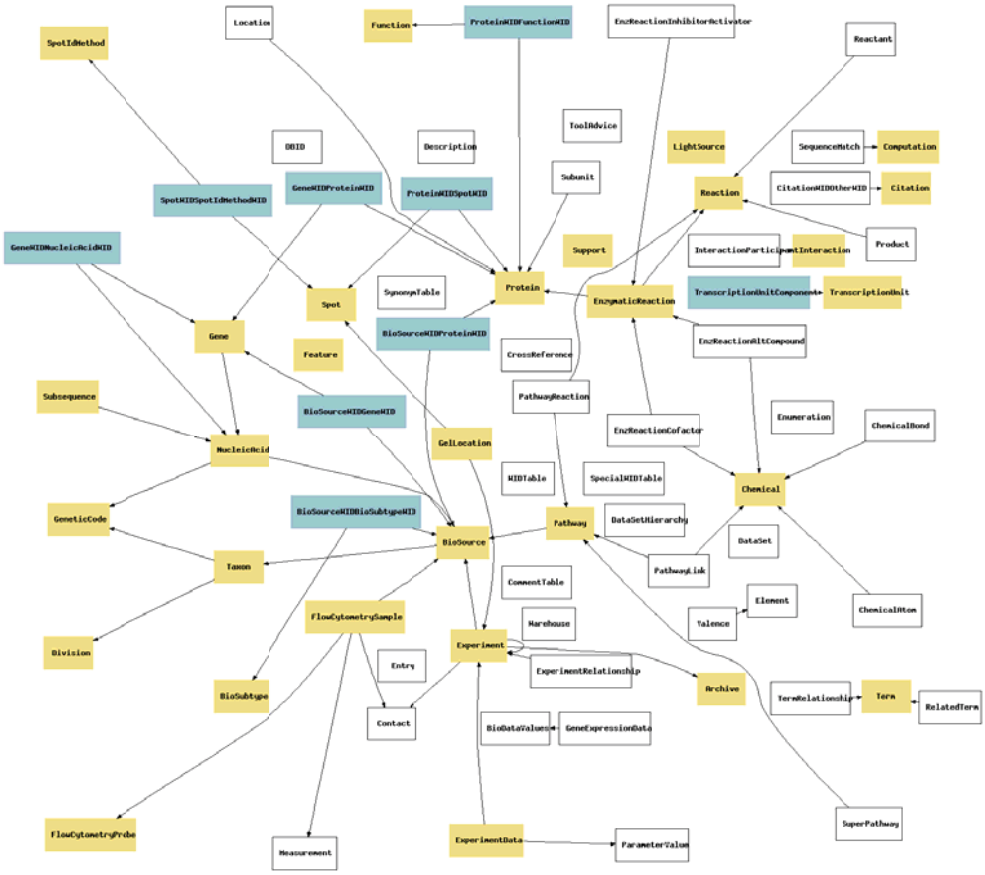

Figure 2.5 - BioWarehouse's database schema

All things considered:

- ✓ Performance
- ✓ Reliability
- ✓ Version control support
- ✓ Large datasets
- ✓ Robust database schema, complex yet usable due to the provided tools
- ✓ Supports MySQL and Oracle DBMS
- ✓ Large community of users
- ✓ Extensive documentation, including developer support
- ✓ Availability (local or remote instances)

- – Steep hardware requirements
- – Maintenance (human interaction required for updating the data)
- – Integration operations suffer from unavoidable long delays in data retrieval and storage
- – Some data sources are yet to be supported (e.g. RefSeq)
- – Certain databases cannot be found in remote, public servers (e.g., KEGG) due to licensing restrictions
- – Microsoft SQL Server DBMS is currently not supported

2.4.2. BN++

BN++ is an open-source biochemical network library and *"a powerful software package for integrating, analyzing, and visualizing biochemical data in the context of networks"* [44].

Developed by the Centre for Bioinformatics Saar and the Centre for Bioinformatics Tübingen, BN++ was designed for biologists and bioinformaticians alike and its purpose is to allow the visualization of complex biochemical processes and networks.

BN++ uses a comprehensible object-oriented data model (called BioCore [45]) that can model nearly all biochemical processes. Built with extensibility in mind, BioCore has two distinct frameworks (in C++ and Java) that enable rapid application development, an important feature for software developers.

From this data model, a data warehouse, that stores biochemical data and its processes, was created using MySQL. Several data loaders that allow data retrieval from about ten different sources (including KEGG, RefSeq [46] and BioCyc [47]) were implemented. During multiple source data integration tasks, some heuristics that prevent data replication have been provided; these allow for controlled, customizable replicated entry removal and play an important part in keeping the database clean and healthy. Nonetheless, manual curation is a still a necessity in these cases. Figure 2.6 (taken from [44]) shows BN++'s architecture:

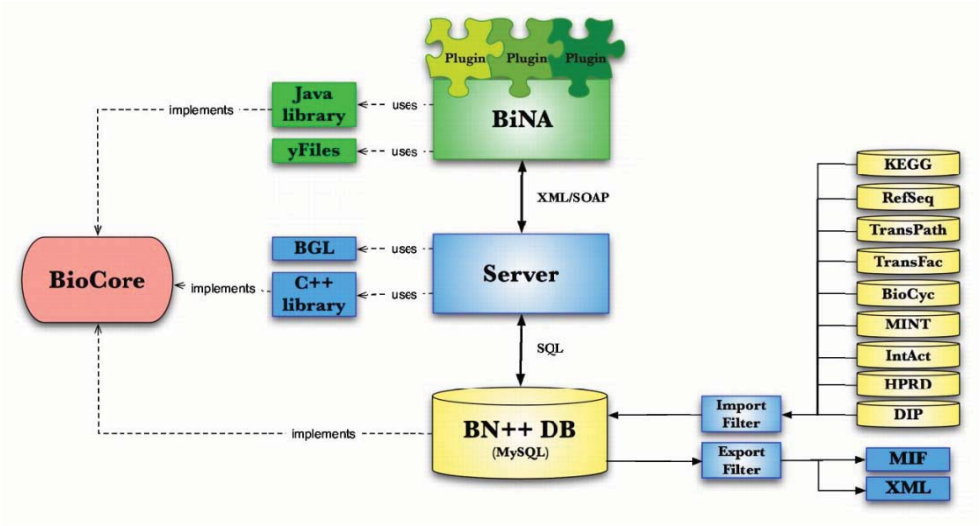

Figure 2.6 - BN++'s architecture

The results can be viewed and analysed through a graphical user-interface called BiNA (Figure 2.7 taken from [44]), developed in Java.

This GUI acts as both front-end to the database and as a data visualization tool. BiNA was carefully designed and excels in usability: the user can browse and manually rearrange items (often multiple in the same graph view) and meta data regarding any object can be immediately retrieved, for example.

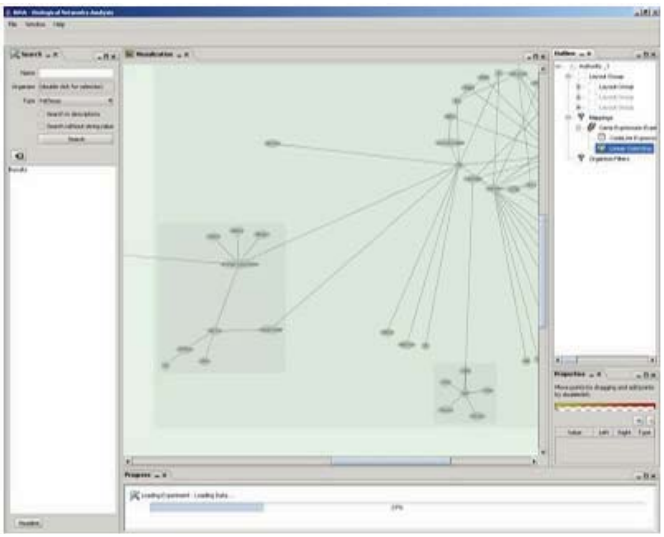

Figure 2.7 - BN++ screenshot

On top of that, BiNA is also an analysis tool due to its graph search algorithms, meta information filters and even a mapping engine that allows the analysis of arbitrary datasets in the context of networks. BiNA also has an included plug-in system that enables support for

modular extensions. BN++ can be ran on a local installation or a remote database (via the BNDB Web Interface) and has a very detailed, eight hundred pages long documentation set.

One can summarize their main advantages and lacks as:

- ✓ Performance
- ✓ Reliability
- ✓ Usability
- ✓ Includes analytical tools
- ✓ Robust database schema
- ✓ Very extensive documentation
- ✓ Availability (local or remote instances)

- – Biochemical networks only
- – Steep hardware requirements
- – Maintenance (human interaction required for updating the data)
- – Integration operations are slow
- – Some data sources are yet to be supported (e.g. Ensembl)
- – Only MySQL is currently supported
- – No updates since 2006

2.4.3. BioMediator

BioMediator is "a data integration system that provides a common interface to Web-accessible sources of biologic information" developed at the University of Washington [35] [48]. As its name suggests, BioMediator is based on the mediator architecture and presets unified data from multiple external sources to its users. This system possesses several variations that confer significant advantages when compared to the more traditional mediator-based approaches. Figure 2.8, taken from [48], shows BioMediator's architecture:

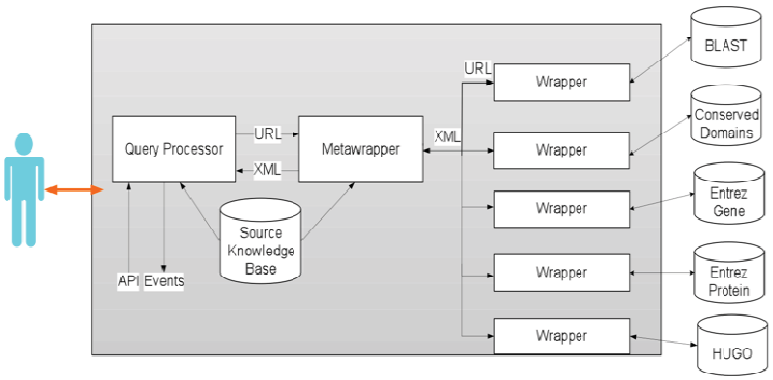

Figure 2.8 - BioMediator's architecture

The modular approach taken by BioMediator bestows the system with a high degree of flexibility and maintainability. By conceptually dividing the required tasks, developers were

able to implement a two-tier model that retrieves and translates data. BioMediator uses four independent components: the query processor, the Source Knowledge Base (SKB), the metawrapper and a set of wrappers. BioMediator's query processor provides an API that allows users to communicate with the rest of the system via URL parameter passing.

The metawrapper, developed in Java, is responsible for semantically transform incoming queries and outgoing results according to the mapping rules residing in the Source Knowledge Base (SKB). This component is the heart of the system and acts as a broker between the query formulator and the wrapper responsible for retrieving the results.

BioMediator does not rely on the creation of views or complex queries that tend to slow down these systems. In lieu, a simplified ontology that contains only the shared entities from each source was developed which, in turn, acts as a basis for the creation of a mediated schema. Because of the ontology it is based on, this schema inherently provides content filtering as undesired sources were filtered *at birth*. This mediated schema is stored on the system's Source Knowledge Base (implemented in the open-source Protégé Framework), along with all the possible data sources, their respective elements and the mapping rules. Due to this module's importance, the SKB must be used locally as a set of class libraries that the users can use to modify the modeled schema via Protégé's GUI. Finally, independent data source specific wrappers (also implemented in Java) allow the metawrapper to connect to the external sources, to pose queries and to retrieve the desired data (in XML format). Subsequently, these results will be integrated into a single XML document in the metawrapper component and returned to the user.

This modular approach renders BioMediator a very flexible and extensible system: the mediation schema can easily be altered; the independent wrappers make the process of adding and updating sources trivial; several instances of the same component may run at the same time in a given computer. All in all, it is a solid system designed with the user's exploration search behavior in mind.

Summarising:

- ✓ Data is always updated
- ✓ Low hardware requirements
- ✓ Modular, extensible architecture
- ✓ User-friendly and efficient

- – Reliability
- – Maintenance (human interaction required for updating the data)
- – Lacks data analysis tools

2.4.4. Biozon

Biozon is *"a unified biological database that integrates heterogeneous data types such as proteins, structures, domain families, protein-protein interactions and cellular pathways, and establishes the relations between them"* [38]. Although originally develop in the Department of Computer Science in Cornell University, Biozon has moved to Stanford University since late 2007.

Biozon implemented a data warehouse system that holds a large number of entries and relationships in a complex, detailed and tight graph schema (Figure 2.9 taken from [38]) built around a hierarchical ontology. In its hybrid schema, each type of data (called a *document*) corresponds to a node and its relationships to edges.

Every type of data can be broken down into smaller subsets (for example, protein sequences into amino acids), a fact that clearly demonstrates the system's high granularity. The system's documents are associated in knowledge domains by means of a hierarchical classification system that orders them according to their origin and content. As a result, each document in the graph is grouped in classes that can relate to their parent classes though an inheritance association. Similarly, a relational classification hierarchy was developed in order to ascertain exactly how two given documents are related; each relation has an associated class that describes its semantics.

The use of these three systems (graph model, document and relation hierarchy) confers a high degree of versatility to the system: Biozon is able to correctly characterize both the global structure of interrelated data and the nature of each data entity, both currently and in the future. Storing derived data is also possible with this implementation, as is running graph search algorithms such as A* or Dijkstra – an important feature unique to this architecture.

Biozon incorporates data from UniProt, RefSeq, PDB, KEGG and GO, among others sources via set of data loaders written in C. During integration procedures, every source's data is converted to a new graph and compared with the current schema, in order to keep data replication to a minimum. This process is also used during database updating operations.

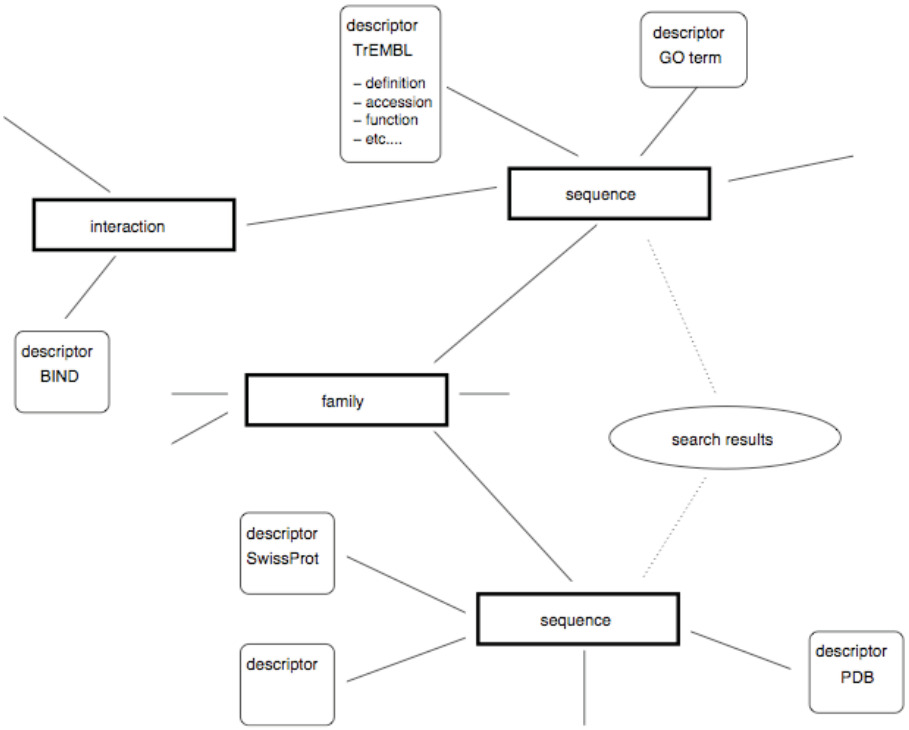

Figure 2.9 - Biozon's schema [38]

Concluding:

- ✓ Performance
- ✓ Reliability
- ✓ Thorough, expansible, non-redundant and coherent architecture
- ✓ Post-integration data processing and analysis algorithms
- ✓ Web interface: http://biozon.org

- – Complex schema makes it hard to develop new applications
- – Steep hardware resources
- – DBMS: PostgreSQL only
- – The DB is not available for download

2.4.5. BioMart

BioMart, formerly known as EnsMart [49], is "a query-oriented data management system developed jointly by the Ontario Institute for Cancer Research (OICR) and the European Bioinformatics Institute (EBI)" [26].

While an open-source solution, this cross-platform system provides access to a large set of data through a series of query interfaces. Despite the fact that, architecturally speaking, an instance of BioMart is a data warehousing solution, this system offers a unique set of features that make

it an excellent choice for the task at hand since BioMart was developed to work with multiple instances of itself. As a whole, it is a set of distributed instances that communicate amongst themselves.

Hence, BioMart can be used as both data source and a Web portal that retrieves data from other sources of data (either local or remote). This unique implementation issues a wide-variety of possibilities, such as enabling the development of domain-specific databases, which can be queried and linked to other instances (if they possess a common identifier). In addition, this system also supports other third-party data sources due to its integration in Taverna [50], Cytoscape [51] and BioConductor [52] (among others), further expanding the range of the data sets.

BioMart uses a very simple three-tier architecture that confers modularity to the system. Firstly, the data integration module: composed of one or more databases (where each database may contain multiple "marts") supporting MySQL, Postgres SQL and Oracle DBMS, this module performs data storage under a reversed star model [49].

The second tier is the Perl API that bridges the gap between the configured datasets and the available data sources. Without this API, BioMart instances would not be able to communicate with each other and, therefore, the system would no longer be distributed.

Finally, the third tier implements the four query interfaces that BioMart yields: MartView, MartService, MartServiceSOAP and MartURLAccess. MartView (Figure 2.10) is the name of the Web interface that enables users to pose complex queries and to retrieve the results, either by displaying them on the website or by downloading them in a compressed file format. In addition, MartView also allows URL/XML requests which can either be constructed programmatically or via the Web interface itself. For the latter, one must click the "XML" button on the top right corner after building the query, copy the results and paste them in the following URL: http://www.biomart.org/biomart/martservice?query=XML[3].

Finally, the results can either be viewed directly on the browser or downloaded via downloader applications (e.g. wget).

[3] The user must replace "XML" with the previously obtained XML code.

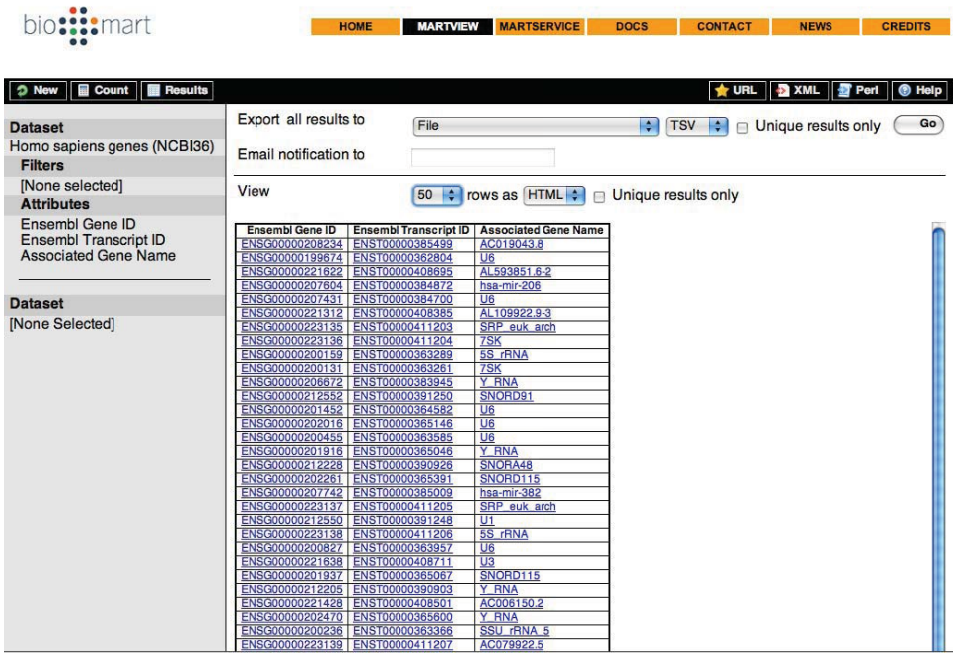

Figure 2.10 – Using MartView to obtain all the associate gene names for *Homo Sapiens*

While MartService is BioMart's REST Web Services layer, MartServiceSOAP is its WS-I fully compliant (hence, with a valid WSDL description file) SOAP equivalent. Both services are also available inside MartView and the syntax for the XML requests can easily be learned by selecting the "XML" button on the top right corner of MartView, after building the request in the Web interface. In the same manner, learning the syntax for the Perl API can easily be achieved by clicking the "Perl" button.

Adding it all up:

- ✓ Unique distributed architecture offers a myriad of possibilities
- ✓ Accessibility
- ✓ Allows the use of non-Biomart sources
- ✓ Easy to set up

- – Performance and reliability (MartView)
- – Steep hardware resources (local installation)

2.4.6. TAMBIS

The TAMBIS [53] Project was developed in 1996 in the University of Manchester and introduced a groundbreaking new feature: Natural Language Processing, that allowed its users to pose queries in human readable language. TAMBIS was a mediator-based Java applet that

had two *modus operandi*: linked and unlinked. Figure 2.11, taken from [53], shows TAMBIS' architecture:

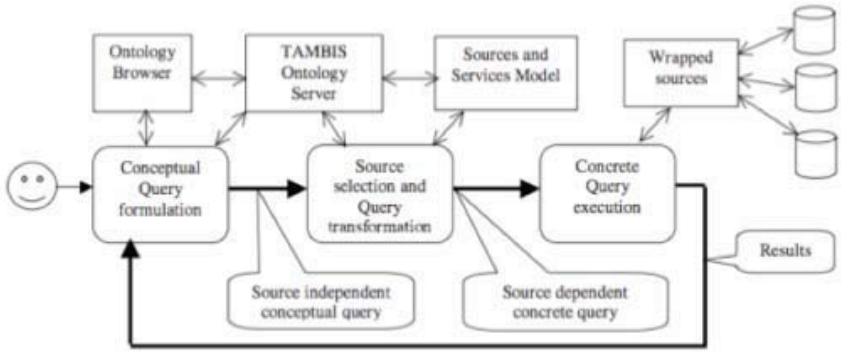

Figure 2.11 - TAMBIS' architecture

While the former enabled querying and browsing remote sources (but its terminology was restricted to 250 concepts), the latter supported around a thousand and eight hundred concepts but could only access local resources.

Unfortunately, the project eventually ran out of funding and ceased its activities. Nevertheless, TAMBIS is still historically relevant due to its innovative, intuitive interface.

As such, the following table presents an overview of the TAMBIS project:

- ✓ Data is always updated
- ✓ Low hardware requirements
- ✓ OS independent

- – The project ran out of funding
- – Performance and reliabilty
- – Limited data sources

2.5. Summary

Mapping numerous biological concepts and the relationships between themselves – whilst overcoming their typically large volume – is a difficult task for which no consensual solution exists. While several distinct architectural approaches exist, each has its own set of advantages and disadvantages; as such, the requirements for the problem at hand should be carefully analysed in order to choose the most appropriate architecture, as should be third-party data integration tools.

Chapter 3 – The GeNS platform

This chapter contains a detailed overview of the GeNS platform in all its aspects, along with a comparison of the previous work.

3.1. Previous Work

GeNS is based on a previously existing biological data integration platform codenamed BioPortal, also developed in the UA.PT Bioinformatics groups. BioPortal attempted to combine the best characteristics of all three data integration methods (data warehouses, mediators and link-based systems), in order to boost the system's performance and user-friendliness. To do so, BioPortal relied on a flexible and reconfigurable architecture that could access data by means of local integration of selected data sets (e.g. Entrez Gene, Gene Ontology), mediated queries (e.g. KEGG Gene, UniProt) and by providing link-based navigation to third party data sources (e.g. KEGG Orthology, KEGG, Pathway, ArrayExpress), according its nature. Figure 3.1 describes BioPortal's proposed architecture.

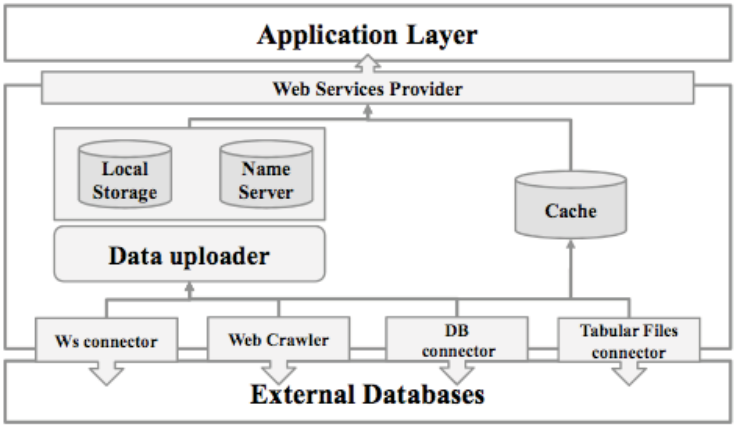

Figure 3.1 - BioPortal's architecture

BioPortal's architecture describes the use of four distinct database loaders – a concept still present in GeNS, as is its local storage component and Web Services provider.

As mentioned, the data would either incorporated in its local storage database or retrieved in real time, according to its nature: large, frequently accessed data sets should be locally stored in order to minimize the system's delay, while other smaller, less used data sets should be either mediated or made accessible via links. If the data was truly relevant (i.e. critical, frequently

accessed data) it would be integrated in the local database. Therefore, it would require pre-integration processing. This processing was, in fact, dynamically performed, as an external XML file (containing the transformation rules) would be accessed, its contents read and applied. As a consequence, adding new data to the system would prove much easier due to the added flexibility of only having to update the reference file. Likewise, it would be very easy to alter the way the data would be mapped in the database.

If, on the other hand, the data were to be retrieved remotely, then the system would have to know all the possible identifiers of a given object in the database (as these tend to have multiple names, according to the sources). To do so, BioPortal required a nameserver that performed that task. This component is, in fact, a database dedicated to the single task of converting identifiers. As an auxiliary performance booster, a cache of the retrieved data would be kept in order to accelerate BioPortal's latency by keeping a copy of recently performed requests. If a user were to repeat a request, there would be no need to contact the third-party data source because BioPortal would still have the results. Thus, the cache would provide a copy of them and the system's latency would be minimal.

Finally, the data would be made accessible by REST based Web Services that could present the results either in XML or JSON format.

However, BioPortal had a design flaw that effectively crippled the system: scalability.

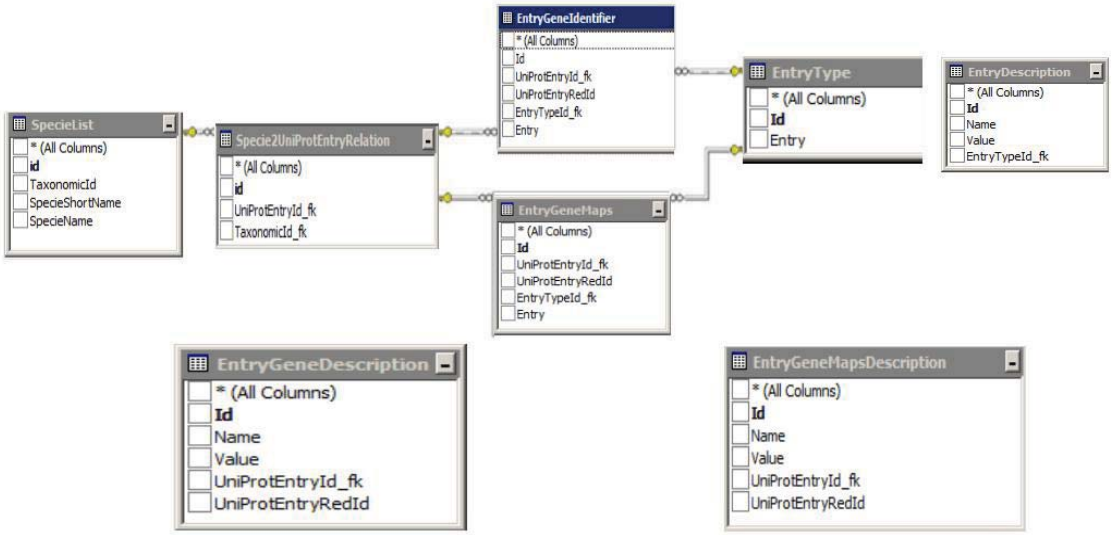

Figure 3.2 - BioPortal's physical database schema

BioPortal's database uses a protein-based hierarchical model. Each organism in the database's taxonomical table has a list of associated proteins, each of them belonging to a single organism. The database followed an OLAP approach, as it hierarchically stored data on a centralized, flake-inspired schema. This hierarchical method of organization not only simplifies the database schema (thus making it easier to understand and maintain) but also greatly improves the system's performance upon queries.

The SpeciesList table is at the root of the schema, storing specie related data. Each species has an unlimited number of proteins stored in the Specie2UniProtEntryRelation table, which, in turn, can have multiple identifiers (in the EntryGeneIdentifier table) and multiple mappings to other biological entities (e.g. pathways). Every entry associated with a protein must have a referring EntryType Id (that keeps track of the kind of data) and may have a corresponding description.

Despite having a low number of tables, this schema can efficiently store multiple kinds of data. However, it is not very easy to understand (mostly due to the names of the tables), nor will it avoid data replication issues with large sets of data because of its low data normalization. If the same protein could be found on a multitude of species (a common occurrence in biology), several distinct entries of that same protein would be created on our protein table (one for each specie on which that protein could be found). For example, if protein X were associated with species Y and Z, there would be two distinct entries for protein X on the protein table. This is a big problem due to the fact that each protein has two different types of relation - one to its protein identifier and another to the related biological entities (which account for large bulks of data). Consequently, adding a new protein to the system could imply the copy the associated biological processes from one protein to another even though these might contain exactly the same data, wasting resources.

This implementation would directly translate in serious data replication. On a long enough timeline, the sheer size of the database would make the system rather useless, making it virtually impossible to add or update any significant amount of data thus seriously crippling its scalability.

In addition, its nameserver relied on a one-to-one mapping structure that made each pair of concepts to be mapped on a table generated during runtime. Hence, if a user wanted to get a list of all the PubMed entries regarding all the genes belonging to the *Saccharomyces cerevisiae* organism (commonly known as Baker's yeast), for example, each gene identifier would have to be stored inside a table called *sce_gene2Pubmed*.

Given the complexity of biological entities, their relationships and the number of third-party data sources, any query that passed through the nameserver would create a large number of tables during its execution, making it impossible for the nameserver to efficiently store and retrieve the required data due to its own size. Simply put, the nameserver was also prone to scalability issues – probably even larger than the ones affecting the database - and, as such, so was BioPortal.

As a result, the platform was placed on hold and only the local storage and its related loaders were implemented.

3.2. Requirements

In order for GeNS be usable, one of the main requirements was that its schema should be **easy to understand and maintain**. To address this issue, we have focused many of our efforts to achieve a comprehensible schema, with a limited number of tables.

Another vital requirement was that the system **should be scalable in size**, in order to contain several gigabytes of data and hundreds of millions of biological entities relationships. The system should also be **scalable in terms of the number of databases** that it stores. This should be obtained without having any changes in the schema. Even containing a huge leap of data, the system **should be efficient in order to give short response times to the most typical queries**. This is especially important because we want this tool to be used to answer user-defined queries and also to be a platform that could be used by other software tools. To attain this requirement, we have stored the gene identifiers and the bio entity entries in separated tables and have optimized the database with the addition of indexes.

The data stored in **the database should be accessible through the use of several methods**. To achieve this we have implemented a set of web services, which can be used to query and extract data from the database, in addition to SQL queries.

In addition, the possibility to **track the current version** of the inserted data, as well as being able to update the existent data without having to change the entire database are two interesting features that should be achieved.

3.3. Architecture

Guided by the previously mentioned project requirements and priotizing query response times, scalability, coherency – and due to the particularities of semi-structured data – the following architecture was defined (Figure 3.3):

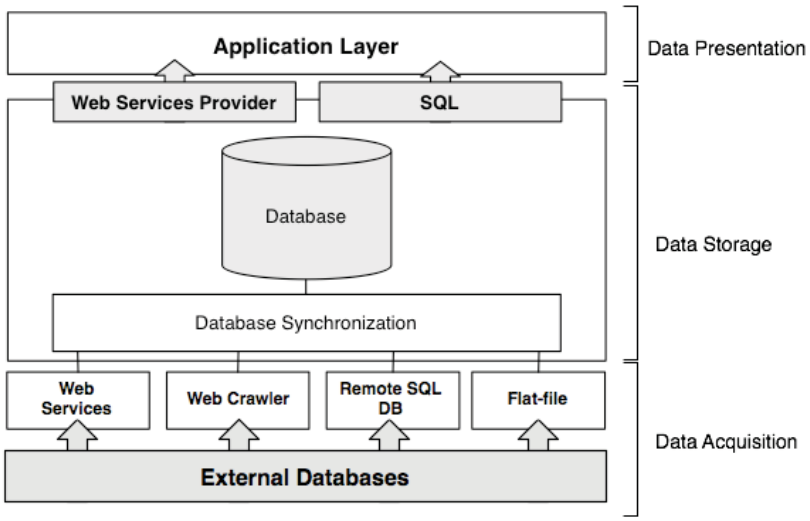

Figure 3.3 - GeNS' architecture

GeNS uses a three-tier, database centric architecture that separates data acquisition, data integration and storage, and data presentation into distinct, non-overlapping processes.

This separation translates into system-wide modularity and data encapsulation (thus implementing Dijkstra's separation of concerns [54]). While the former trait presents improved maintainability, faster development and flexibility by dividing the tasks into modules, the latter increases the system's stability by providing well-defined, solid interfaces that maintain the consistency of the system (even after changes in the lower layers) and restrict the user's queries to a set of pre-defined methods (thus, increasing the system's security).

GeNS eschewed the nameserver approach that flawed BioPortal. Instead, the data warehouse facet rose to prominency by storing a larger set of types of data and uniting them under an internal identifier, ensuring their consistency.

In the following sections, each tier of the model will be extensively described.

3.4. Data Acquisition Tier

As its name clearly indicates, the Data Acquisition tier encompasses data retrieval and transformation operations. Owing to the fact that biological data sets are spread across several databases - each with its own schema and accessibility options – methods to retrieve the desired information had to be developed, in order to populate GeNS' database.

About the data sources

Selecting the most representative external data sources and determining how their data can be retrieved are the first two steps towards providing a unified view of the information. Any external data source will only be suitable for GeNS if these two very important conditions are met.

Regarding the former, selecting the correct sources goes a long way in determining the overall quality –and usefulness – of the data integration system. The sources should provide interesting data sets (preferably large, curated and of high-quality), as the system would surely lose its usefulness if its data were incorrect or rather limited. In addition, a certain level of correlation between the sources must exist in order to successfully map cross-references amongst entries of distinct sources, i.e., the sources must provide references from their entries to correspondent entries in other databases if a unified view of the data is to be created. Typically, molecular biology databases already provide references from their entries to the correspondent ones in other databases, albeit incomplete in both numbers of entries and of sources.

Not surprisingly, correctly correlating entries from different sources is yet another classical problem in this field of work for which no consensual solution has been found [55].

The sources must also present adequate data retrieval methods if the source is to be of any practical use; after all, even with very interesting data sets, a source is still rather useless if one cannot access its data satisfactorily. As seen in chapter 2.2, there are three methods through which external sources can publish their data:

- Flat files
- Web Services
- SQL Access

Flat files are an ideal choice for transferring and importing large chunks of data, as the high-bandwidth of the servers in which these files are stored allows for quick, painless access. When taking into account GeNS's data warehouse based architecture, where the users only have to wait once for the parsing procedures (after which, the data is integrated in the local database thus no longer requiring any transformations), the benefits clearly outweigh the drawbacks. Unfortunately, not all sources choose to publish their data this way (e.g. Biomart), pushing its users towards other options such as Web Services and SQL Access via RDA. Due to the fact that the former is much more common amongst third-party data sources, developing Web Services data loaders was deemed an important objective for GeNS (superseded only by flat-files loaders) and was given priority over SQL Access scripts.

Having identified the main selection requirements, the following data sources were selected: UniProt [19], KEGG [21], OMIM [56], NCBI Taxonomy [57], Biomart [26], ArrayExpress [27], PubMed [58].

UniProt, born from the fusion of Swiss-Prot [59], TrEMBL [59] and PIR [60], is and will be a top pick for any biological data integration system. Containing over a hundred and thirty types of data, it provides a publicly available, very comprehensive and frequently updated set of proteinic data that, simply put, cannot be ignored. Although comprised of several databases, only UniProtKB (UniProt's Protein Knowledgebase) is currently being integrated in GeNS. UniProtKB encompasses two very important sources of data: Swiss-Prot and TrEMBL. While the former provides high-quality, manually curated proteinic annotations, the latter presents a very large set of unreviewed, automatically annotated data.

All in all, over five gigabytes of compressed data - which grow to thirty gigabytes, after uncompressing - are periodically published (more specifically, two XML flat files) in each release of Swiss-Prot and TrEMBL in UniProt's FTP server. UniProt's data must be parsed before integration procedures, in order to filter out unnecessary the XML overhead and unnecessary data.

KEGG (Kyoto Encyclopedia of Genes and Genomes) is another very important source of data that aggregates over nineteen different databases. Presently, GeNS incorporates four of them: KEGG Gene, Orthology, Pathway and Drug.

While the first source publishes its data in tabular formatted Flat files (that do not require parsing before integration procedures), data from the other three sources is in a loosely structured, human recognizable format that must be parsed in order to enable its integration. In

addition, each source has its own content specific format, due to the need to store different fields.

For example, KEGG Drug must store atom-related data, while KEGG Orthology does not due to the nature of the data itself. As a consequence, KEGG Orthology Flat files do not possess the "*ATOM*" tag found in KEGG Drug's. As a result, each source must have its own modified parser, perfectly adapted to the flat file's format.

Despite the fact that these sources already encompass a large variety of concepts (UniProt alone has over a hundred and twenty different types of data), some important concepts are still missing (such as gene locus information and expression data), while others are indeed present yet incomplete (e.g. genetic disorders, literature references, taxonomical details).

NCBI's OMIM [56] (Online Mendelian Inheritance in Man), a thorough, curated human genes and genetic phenotypes database. As previously mentioned, in spite of the fact that UniProt already comprises this kind of data to some level, the possibility of integrating data straight from OMIM bestowed GeNS with a much more detailed level of knowledge. OMIM allows direct data retrieval via its FTP server, where Flat files (with its specific loosely structured, human recognizable text format) containing phenotypes, gene and disease names, among others, can be found.

Similarly, UniProt (more specifically, SwissProt) also possesses a manually selected set of high-quality literature citations. Nonetheless, quality comes at a price: quantity. The need to link more and more articles regarding a given entry in the database is a possibility that should not be ignored. As such, GeNS also integrates data directly from NCBI's PubMed in order to map as much information as possible.

PubMed is a quality search engine for biomedical articles that contains millions of entries and is powered by the Entrez retrieval system. PubMed has several tabular formatted flat files in its FTP server that allow direct gene to article mapping, facilitating its integration in GeNS.

UniProt also provides rather rudimentary taxonomical data (by keeping track of the organism's taxonomical id only); in order to solve this issue, this data was complemented with NCBI Taxonomy data. As a result, the organism's scientific name and short name are now fully integrated in GeNS. In addition, clinical trial data is also being integrated through the Arabella crawler [61]. Using a web crawler enables retrieving data only available on web sites, albeit at a slower pace than via parsing flat-files or even Web Services. After crawling through the U.S.

National Institutes of Health's Clinical Trials website [4], the data is returned in an XML formatted flat-file. This file is subsequently parsed and imported to the database via an SQL script. Finally, the last two chosen sources: Biomart and ArrayExpress both incorporated via Web Services. Biomart was selected due to its extensive gene locus information (that UniProt did not have) and ArrayExpress was because of its expression data (also missing in UniProt). Altogether these databases represent a very healthy set of data that span over 150 different data types. By merging all of this data, almost 7 million unique gene entries and over 120 million biological relationships were obtained.

Figure 3.4 shows the most relevant of GeNS' sources. Some of these are being indirectly integrated, as they are already present in UniProt. Hence, UniProt, KEGG Gene, Pathway, Orthology and Drug, NCBI Taxonomy, PubMed, BioMart, ArrayExpress and OMIM are being directly integrated, while NCBI Entrez, RefSeq, GenBank, ExPASy, PharmGKB, Gene Ontology and InterPro (among others) are not.

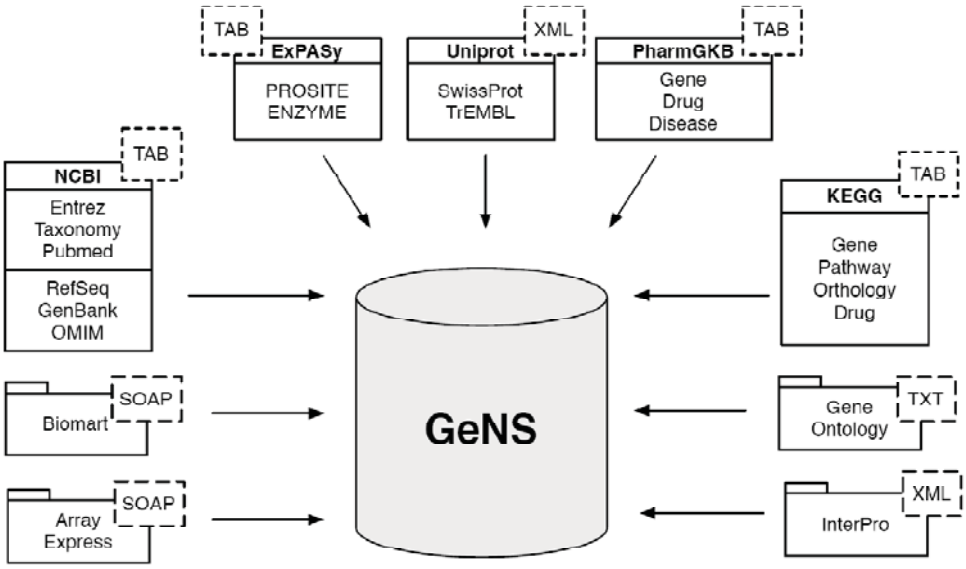

Figure 3.4 - Schematic representation of the databases integrated in GeNS

The data loaders

Having chosen the sources and how their data will be retrieved, multiple flat files loaders - supporting XML (e.g. UniProt, clinical trials), tabular (e.g. KEGG Pathway) and even human-recognizable, loosely structured text files (e.g. KEGG Drug, KEGG Orthology) - were

[4] Available at http://clinicaltrials.gov/

developed, along with two Web Services loaders (e.g. Biomart, ArrayExpress Atlas), in order to enable integration of the desired data sets.

These loaders were implemented as static, independent modules (grouped by data source), used in a single application codenamed BioPortal Db Builder and built in C#. Inside the application, each set of loaders – organized by source (both in the source code and in the graphical user interface) – accesses a number of flat files and parses their content, molding the data to a tabular format capable of direct integration to the local database (using the DBMS). This application can be seen in Figure 3.5; as it was designed for internal use, its GUI was deemed as a secondary objective and, as such, it needs to be polished if the application is to be publicly available.

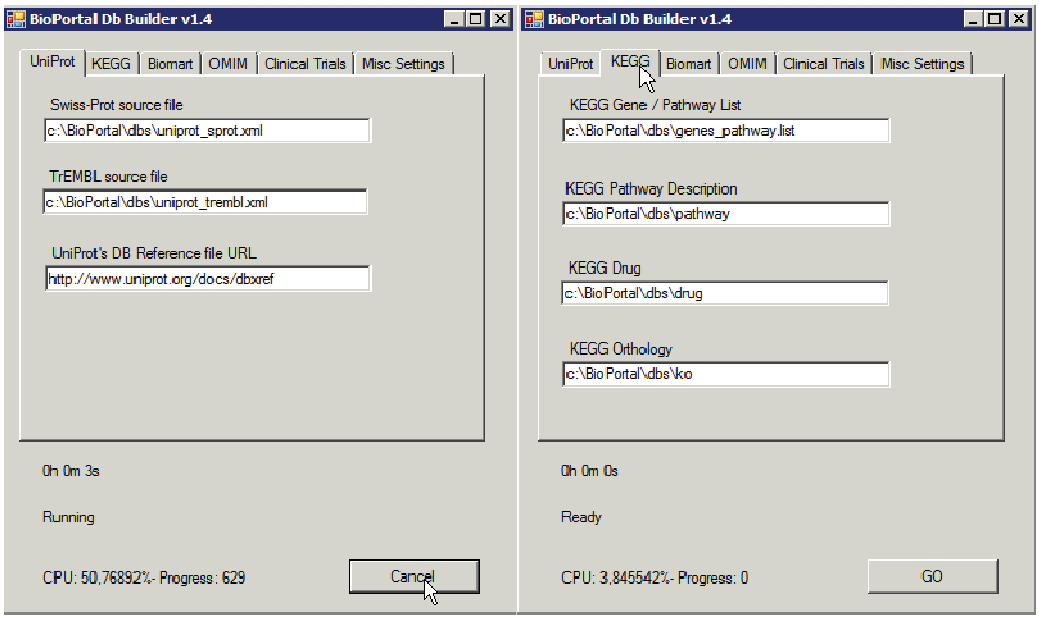

Figure 3.5 - BioPortal Db Builder parsing Swiss-Prot and TrEMBL files, along the interface for KEGG, respectively

The exception is the Web Services loader for ArrayExpress Atlas that was built as a Web Service itself. The reason for which was that this loader began as an experiment to determine how to correctly download data from Atlas. Eventually, the code to develop until it met its objectives and, as a consequence, converting the ArrayExpress Atlas loader to a distinct C# module became an optional objective because the data could still be retrieved.

By implementing each loader as static, independent modules, a rapid development was achieved due to their inherent simplicity and easy integration. The alternative would be the implementation of dynamic modules that would follow the guidelines of an external reference file (e.g. a XML file) to perform the data processing. This approach would grant a high level of

flexibility, making it very easy to cope with changes in the format of external sources: simply update the external reference file correspondingly. Nonetheless, the former option was chosen over the latter due to its greater complexity, which would slow down the development process. Moreover, the format of a source's flat files tends to suffer few changes along the years because of stability purposes, a fact that tipped the balance even further towards static loaders.

This decision proved to be correct as the flat files' format effectively remained the same along the development of this system. Nonetheless, in the uncommon event of a data source changing the format of its flat-files, the code must be reassessed if the data is to be correctly parsed again. In that case, since each loader is, in fact, a module the entire application will not stop importing whichever data it cans, nor should it raise problems in the insertion of the new loader.

The UniProt loader is probably the most important loader of the whole set, because of the sheer amount of data it must process: over thirty gigabytes in two XML files, one for SwissProt and another one for TrEMBL. After parsing the relevant data, the output files (in tabular format) still occupy four gigabytes of data (an amount that surpasses by far the other sources; KEGG Pathway, even before being parsed, is the second largest source and only needs three hundred and fifty megabytes). Hence, performance is a crucial factor if the database is to be built and populated in a reasonable amount of time.

Despite the fact that BioPortal had intended to access UniProt's data by means of a mediator, when it became apparent that the platform could not support its designed architecture (mostly due to its nameserver) the data warehouse approach gained momentum – in order to circumvent the issue – and, as such, UniProt's data now had to be parsed and stored locally.

To do so, a parser was built. This parser is not the currently used one because of two reasons: its performance and reliability. The older UniProt parser took over twelve hours to parse and load the data on the local database and, in some cases, the data contained foreign characters that effectively corrupted its original value.

The reasons behind these issues were twofold: firstly, the results for each entry in the source's XML file were being directly inserted in the database (after checking for duplicates) by an active SQL connection; this meant that every protein in UniProt would require at least one SQL Insert command - in fact, more than one as the insertion would occasionally fail (and four insertion attempts per entry were manually defined in the loader's source code).

This is a major bottleneck to the platform's performance because running multiple insertion statements will always be slower than running a single insert statement with the other (multiple) statement's data. Therefore, the first and foremost optimisation was to use efficient data structures (hash tables) which stored large blocks of parsed data until their manually defined threshold (a hundred thousand entries) was met; then, the data would be written to tabular formatted flat files (thus flushing these structures) and could be imported to the database.

Secondly, the methods used to access, navigate and retrieve the desired information through the XML fields were analysed. At the time, the entire file would be loaded into an *XMLTextReader* class instance and, for each detected entry; an *XMLDocument* class instance would be created. The elements for each entry would then be collected via the *SelectNodes* method, whose output would be used to create *XMLNodeList* class instances that would directly inserted into the database.

After having investigated all the possibilities regarding the task at hand, the *XMLDocument* class was deemed inappropriate due to its poor performance [62] and was replaced by the *XPathNavigator* class due to its combination of query-based access to the data and superior performance.

Having performed these optimisations, a testing period followed. The new UniProt loader took four hours to parse the XML files; even with the addition of an extra hour - required to import the data to the local database (a task performed by Microsoft's SQL Server, GeNS' DBMS) – parsing and integrating UniProt data now only requires five hours instead of the twelve it used to, roughly a **sixty percent** improvement. In addition, the data corruption that plagued the old parser's results from time to time was purged with the arrival of the new parser. The bug was never found, mostly because the new parser was built from scratch.

UniProt releases new updates on a monthly basis and, occasionally, new kinds of data are added to its data set, usually from the integration of new third-party data sources. As such, it is important to check if GeNS already has these new sources referenced. Fortunately, UniProt has a public list[5], which states all the currently cross-referenced third-party sources, that enables this operation in a very simple manner.

---

[5] Available at http://www.uniprot.org/docs/dbxref

As such, every time that a UniProt flat file is ran through the parser, GeNS' list of known data types must be compared to UniProt's list of referenced data sources – downloaded in real-time directly from UniProt to ensure that the file is updated – in order to detect if data from new sources has been added. If that is the case, then GeNS' list will be updated and dumped to a flat file (in tabular format), which will subsequently be imported through SQL scripts. This task – source synchronization - can also be performed without having to parse any UniProt flat-files; to do so, one only has to access the "Misc Settings" tab and click on "Update Data Type CVS File".

Biomart and ArrayExpress Atlas' loaders are also noteworthy; the former works by retrieving the data from MartView's URL/XML requests; the users must select the desired species, gene-related information and a maximum of three external reference identifiers from the GUI depicted in Figure 3.6. Biomart itself imposes this limit and in the event of choosing, for example, four external references an error message will be returned (which, due to the way the parser is built, will be downloaded and stored inside a text file).

Programmatically speaking, each active checkbox has a corresponding XML string that will be concatenated into a single string after clicking the "Get Data" button. Then, the resulting XML string will be passed along to Biomart's MartView, which will validate, process and return the results, which the loader will store in a flat file (one for each species). These tasks are performed by a BackgroundWorker class instance that provides multithreading capabilities to the system. Finally, a progress bar keeps track of the overall progress.

Finally, the ArrayExpress Atlas loader can be access by clicking on "Launch Atlas Retriever". This loader was not implemented as a module, but as a Web Service procedure instead that allows the users to select the output directory, the desired taxonomical organism and, subsequently, to download all the desired expression data.

However, Atlas' API only allows retrieving data by gene. As a consequent, the loader has to open an SQL connection with the database, retrieve all the gene names for the human species and to run them through Atlas in order to get all the experimental data for human genes.

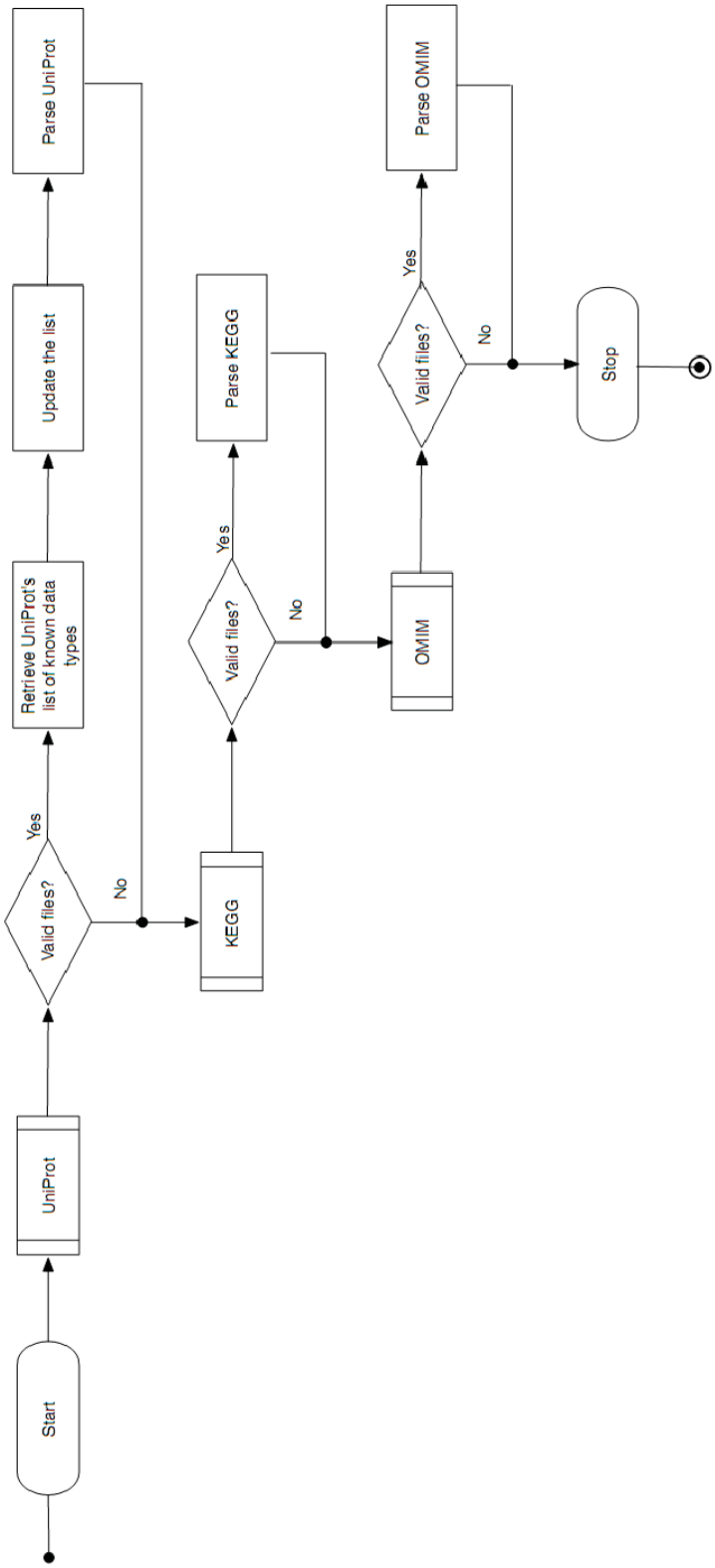

Figure 3.6 - The activity diagram for parsing flat files

In order to speed up the process, this loader breaks the list of genes names into multiple parts and passes them to threaded processes (eight) that collect expression data for each gene. The results are stored in a single hash table and if the number of entries in this structure passes a manually defined threshold (currently at a hundred thousand entries), the hash table is dumped to a file. As this operation (dumping to a file) is not thread safe, a semaphore restricts the thread's access to this shared resource during the dump. As usual, the output file is a plain text, tabular formatted file that can be easily imported into the database.

Regarding PubMed and NCBI Taxonomy, there is no need to parse these results as they already are in a tabular format that the DBMS can directly import.

Finally, SQL queries that allow the DBMS to synchronize the database with the content of these flat files were developed. It takes one hour to import UniProt's data; it is the slowest procedure of all, mostly due to its size. The remaining sources range from fifteen minutes (PubMed) to a matter of seconds (KEGG Gene).

3.5. Data Storage Tier

The data storage tier is responsible for storing data retrieved from multiple sources. To do so, a relational database was created using SQL Server 2008 mainly due to its good trade-off between performance and easiness of use. The schema evolved from BioPortal's database, following OLAP's directives regarding iterational improvements.

Once more, providing a correct representation of biological data without sacrificing the system's performance or scalability, among a long list of requirements, is still a challenge in bioinformatics [14]. In order to address this issue, one of two opposing schema design principles is typically applied: generalization or specialization.

A general schema prioritizes flexibility, scalability and the integration of several types and large volumes of data. It uses a large, dynamic set of data sources (which may vary throughout the time) in order to encompass as much diverse data as possible. A database designed according to this principle will allow its users to correlate heterogeneous data and to, eventually, extract conclusions that would otherwise hardly be visible.

On the other hand, a specialized schema accommodates only a limited number of datasets. Usually, only a handful of sources of data are used: these sources were chosen from the very beginning and usually remain unaltered for long periods of time. This schema is usually

considered as more suited to address more specific issues once that unlike general database schemas, scalability and flexibility are secondary aspects.

Practically speaking, either the schema is adapted to fit the data or the data is adapted to fit the schema; from these two approaches, several intermediate levels can be derived. GeNS manages to take advantage of both methods, adapting both schema and data to one another: to physically store the data a general schema that certifies the scalability and flexibility of the database is used, drawing strength from a concrete meta-model where all the entities and relationships are specified.

3.5.1. Meta-model design

GeNS' meta-model is essentially gene-centric because, biologically speaking, it is the simplest and most effective way of mapping the vast majority of concepts and their relationships together.

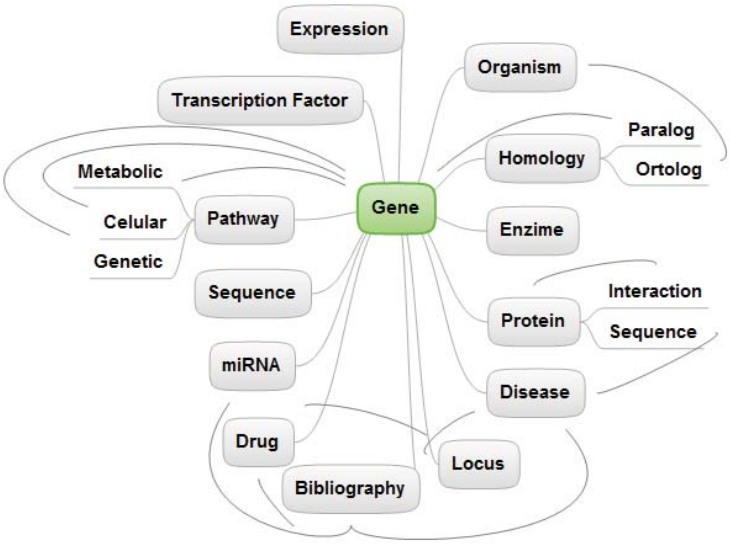

Figure 3.7 - GeNS' meta-model

As one can see in Figure 3.7, related with each gene there is a network of data types that can be directly associated with it, such as pathways, transcription factors, proteins, etc. These data types would, otherwise, hardly be mappable between themselves. Adding new kinds of data only requires changing the meta-model, not the physical schema.

3.5.2. Physical schema design

Despite the fact that on a conceptual level a gene-centric approach is the most logical choice, the physical model is protein-centric (Figure 3.8) instead because of the difficulties encountered in implementing the gene as the "heart" of the database. Biological peculiarities, in the end, rendered a gene-centric physical schema unviable.

For example, *Homo Sapiens* has over twenty five thousand unique genes. However, some genes can be duplicated in the genome – paralogous genes – thus occupying two different chromosomal locations in the same organism. Paralogous genes tend to share the same function, but in some cases the copy may have altered its original function or even picked up a new one during a mutation or even from simply having swapped its position (due to possible interactions with other codons); as such, these genes can be functionally different. In fact, this process is so important that many authors believe that it is the underlying factor that powers evolution [63] [64].

In this particular case, the same gene could be synthezing different proteins – how could these proteins be accurately represented in the database if the same (gene) identifier was responsible for generating two very different sets of results? By adding another piece of information to the gene identifier, such as its position within the genome (*locus*) for example.

Therefore, using the gene as the basis for the physical schema is not a good choice because of the complications derived from their very nature. Instead, proteins make a better basis for physical schemas because they are biologically a more coherent choice, less prone to peculiarities. Since each gene synthesizes a number of proteins, the relation between gene and proteins can easily be stored inside the database.

This direct protein-gene relation also enables mapping the remaining concepts seen in the meta-model; when retrieving a gene related concept from the database, the gene identifier is looked up, translated into the corresponding protein identifiers and the desired data can now be easily accessed. This way, all the inherent biological issues of using gene-centric physical schemas are avoided and the solid gene-centric meta-model can still be used.

Although it might make sense to swap the metal-model to proteins in order to match the physical model, certain concepts cannot easily be associated with proteins instead of genes. OMIM, for example, contains a catalog of human genes and genetic disorders. As expected, this kind of data has a direct relation with genes and swapping it relation for proteins would not

make the conceptual model easy to understand. As such, it is preferable to maintain the current meta-model.The database's tables will now be explained:

- **Organism**: Stores taxonomic information; each entry corresponds to an organism with any given number of associated proteins. This table is the root of the hierarchical model. For each organism, we store organism detailed information such as its scientific names and reference sequence.
- **Protein**: This table stores information regarding each protein entry. This information includes gene locus, gene and protein sequence and the relation to two distinct tables: Identifier and BioEntity.
- **Identifier**: Contains all the synonyms, alternative names and identifiers for each entry.
- **BioEntity**: This table stores unique identifiers belonging to the biological entities associated with a given protein; this includes, among other things, pathway, gene ontology and gene expression identifiers. Detailed data regarding a specific entry in this table will reside in the Description table.
- **BioEntityDescription**: The description table stores structured data related to a specific biological entity. Examples of usage include the detailed description of a pathway or the mutation of a genetic disease.
- **DataType**: Contains a list with all the types of data retrieved from external databases, encompassing both identifiers and biological entities. Every entry in either Identifier and/or BioEntity tables references this table, so that we may easily determine the type of the data, thus preventing semantic related errors (e.g. comparing two completely unrelated objects).

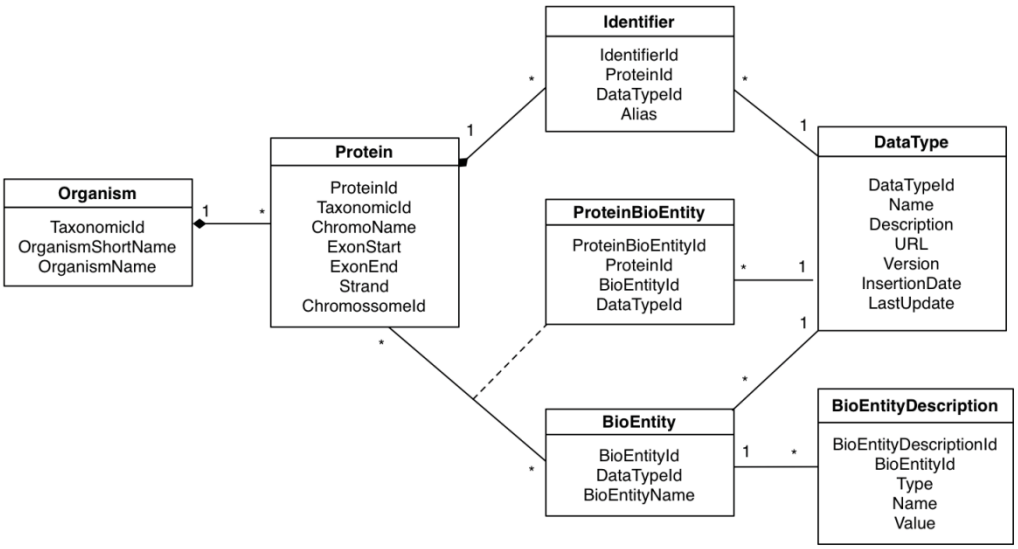

Figure 3.8 - GeNS' physical schema

GeNS' physical schema is an improvement over BioPortal's mainly because of its intuitive, user-friendly OLAP-like database and due to its new improved mapping system between protein and biological entities (e.g. pathways).

Regarding the first, nearly all tables and their respective fields were renamed; as such, they now possess meaningful, easily memorizable names. When combined with the database's simple yet effective structure, developing applications that run on top of GeNS becomes a quick and painless task, as developers are able to focus on the visualization and analysis levels instead of the database itself.

The former issue was also quite relevant for BioPortal, as it ensured that the system would not be usable given a long enough period of time. As mentioned, BioPortal's physical schema did not store relationships, but biological entities instead. Each biological entity associated with a protein would have to be stored locally, thus increasing the level of data replication.

The answer was to increase the level of data normalization: GeNS stores associations between biological entities and proteins instead of the actual entities. Each biological entity in the BioEntity table is unique and, thanks to a correlation table, multiple to multiple connections between proteins and biological entities keep data replication down to a bare minimum, while ensuring that the system's scalability and performance remain unaffected, along with all the benefits provided by the hierarchical model.

3.5.3. Exemplifying

The following example demonstrates one of many possible scenarios in GeNS: a researcher wants to obtain the network of concepts related with the gene: 'sce:Q0085' (Figure 3.9).

The system starts by determining the internal protein identifier through the Identifier table. With this identifier, we can now determine the alternative gene identifiers (still within the Identifier table).

Subsequently, the system will ascertain the corresponding organism; in this particular case, we already know the answer due to the first three letters of the identifier (sce, the short name for Saccharomyces cerevisiae) but this fact will not affect the process. In order to do so, GeNS looks up the Protein table and uses the taxonomic id to identify the organism in the Organism table. In the Protein table it is also possible to find the gene locus, its sequence and a general description.

Following this procedure, GeNS maps every biological entity associated to our pre-determined protein identifier by looking up the ProteinBioEntity table (that contains all the relationships between the two). This allows GeNS to retrieve the biological entities in the BioEntity table which, in turn, contains homology, bibliography, expression, ontology, pathway and enzyme related data, among others. Finally, more details about each biological entity can be obtained by looking up its description in the BioEntityDescription table.

Extending this example, the researcher wants to obtain all others genes related with the KEGG pathway 'sce00190' where the gene 'sce:Q0085' was initially present. To do so, he searches the Protein table for all the entries that contain a relation to the table BioEntity that matches the required pathway.

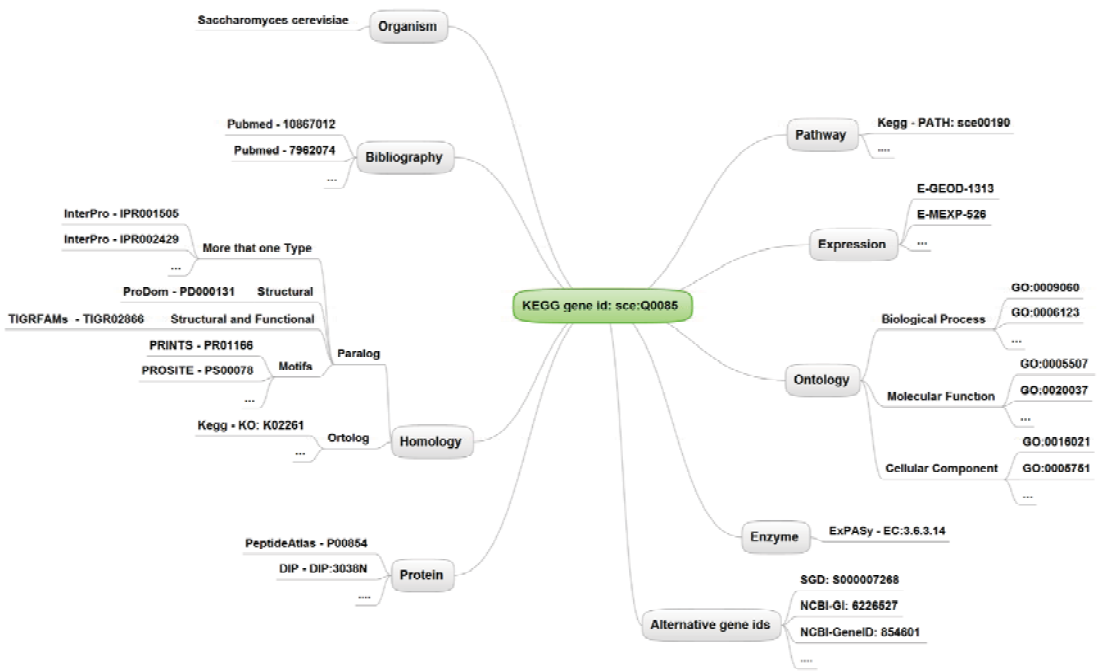

Figure 3.9 - An example of a typical query to obtain the network of concepts related with the gene 'sce:Q0085'.

Moving along to the SQL queries - if another researcher wanted to find out which genes are associated with the pathway '*path:aae00010*', for example, then the code would be:

SELECT DISTINCT Alias FROM Identifier where DataTypeId = 1 and ProteinId in (SELECT ProteinId from ProteinBioEntity WHERE BioEntityId in (SELECT BioEntityId from BioEntity where BioEntityName like 'path:aae00010'))

Likewise, if the task at hand were to retrieve all the known synonyms for a given protein (e.g. HLA-B), the query would be:

SELECT DISTINCT Alias, DataTypeId FROM Identifier WHERE ProteinId in (SELECT ProteinId FROM Identifier where Alias like 'HLA-B')

3.5.4. Loading the data

As mentioned, a common issue when integrating heterogeneous datasets consists in correctly correlating them [13]. This is particularly important in GeNS because of the large number of sources, where new data coming from multiple sources must be correctly mapped to the corresponding entry in the GeNS' database; the identifiers from the new datasets must have a direct match with the identifiers already present in local database, in order to ensure that the new information is correctly associated with the right objects.

Not surprisingly, after retrieving and parsing the desired data sets (in the previous tier), the resulting data must be loaded via SQL scripts in any order, provided that UniProt's is the first and foremost. This is because of UniProt's excellent identifier coverage, referencing over a hundred distinct objects thus providing a solid base for directly mapping objects from other dataset to another.

3.5.5. Cross database mapping

Although molecular biology databases already provide references from their entries to correspondent entries in other databases, these references tend to be incomplete both in number of entries as well as in the number of databases covered.

To address this, a methodology that spans a tree of alternative paths between two databases was explored. The idea is to look for alternative identifiers in other sources of data that may already have a direct correspondence with the identifier(s) in the desired database. To do so, an algorithm that attempts to retrieve corresponding entries in other databases was developed (Figure 3.10).

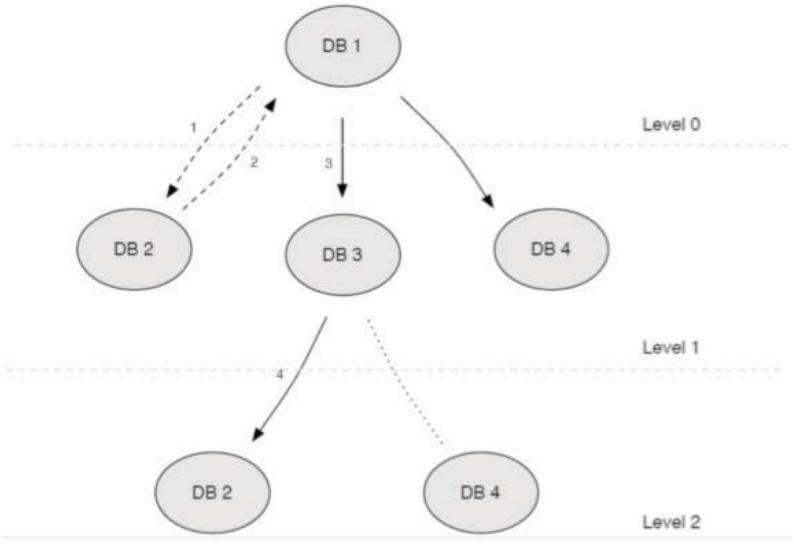

Figure 3.10 – Schematic representation of the cross database mapping algorithm used to improve the coverage of databases.

Considering each database as a node, and each database reference as a connection, a digraph that represents real linkage across databases was created for each entry. The first step of the algorithm consists in verifying the existence of a direct connection from DB1 to DB2. In case it fails, the next step consists in testing the reverse connection, as DB2 may hold a correspondence between the identifiers even if DB1 did not. If it also fails, the list of directly connected nodes is obtained, and, for each, the same procedure is followed. This procedure can be seen in step 3 of the example shown in Figure 3.10.

Hence, DB3 is the next tested node after checking that neither DB1 nor DB2 had a direct correspondence. As DB3 contains a direct link to DB2, the algorithm finalizes by returning this value to DB1. The algorithm stops after having found a valid path, skipping DB4 altogether because a direct match between the identifiers has been established.

If the algorithm were to continue, it would have to transverse the entire graph to search the deepest node and, subsequently, to backtrack it in order to follow all the available nodes. This behavior is not required because, as mentioned, by being able to establish a direct connection between identifiers the new data can now be inserted, thus avoiding the need to look for further correspondences. The complexity of the task at hand is directly related with the number of available sources, hence selecting a handful of sources that contain good identifier coverage between themselves is heavily recommended, as the algorithm already requires several hours even with a relatively small number of sources.

In GeNS, this method is currently not yet implemented because it was superseded by other tasks (e.g. developing the Web Services layer). Nevertheless, an experiment was made in order to determine its usefulness.

Four of the most relevant databases present in GeNS were selected: UniProt, KEGG Gene, Ensembl and Entrez Gene. These four databases store links to more than one hundred and twenty distinct biological entities, proving a very good coverage regarding identifiers. In addition, this analysis was restricted to the *Homo Sapiens* organism, in order to make it as simple as possible.

The initial step consisted in measuring the coverage - the percentage of entries in the origin database that have direct and explicit correspondence in the target database - value of each database. To prevent tainting the results, no inference was performed in the calculations (such as knowing that, for the human species, concatenating "hsa:" with the Entrez gene identifier will lead to the KEGG identifier, for example).

Table 3.1 - Comparison of the average coverage values with and without the use of the cross database mapping algorithm.

	UniProt		**KEGG Gene**		**Ensembl**		**Entrez Gene**	
TO / FROM	**Before**	**After**	**Before**	**After**	**Before**	**After**	**Before**	**After**
UniProt	-	-	83,8%	83,9%	97,1%	97,4%	85,7%	**93,1%**
KEGG	76,8%	**80,5%**	-	-	77,2%	79,8%	100%	100%
Ensembl	97,7%	97,9%	0%	**73,2%**	-	-	88,9%	89,2%
Entrez Gene	47%	**57,4%**	0%	**34,3%**	46,8%	**57%**	-	-

Table 3.1 shows the coverage values obtained before and after running the algorithm. A significant improvement in the coverage has been obtained in pratically every relation. Entrez Gene in particular saw its coverage to other databases significantly increased, essentially due to its initial low values; Entrez Gene to UniProt has noticed a positive difference of 10,4% and to Ensembl 10,2%, along with a 34,3% increase to KEGG Gene. Other relevant improvements have been registered for the Ensembl to KEGG Gene relation (73,2%), KEGG to UniProt relation (3,7%) and for the UniProt to Entrez Gene relation (7,7%).

Therefore, the cross database mapping algorithm is a valid method to increase the coverage of the system and further efforts should be made towards its implementation. Ideally, this

algorithm should be executed after the initial UniProt integration but before other data sets, in order to maximize the identifier coverage by attempting to map unknown identifiers, thus attempting to integrate as much data as possible.

Finally, although this method is currently only used to assure the maximum coverage of the stored identifiers with minor changes it could also be used to detect and remove mismatched and overlapping identifiers.

3.5.6. Optimizing the database

Providing an efficient physical schema was only the first step towards building a high-performance data integration system. Given the large volume of the integrated data sets, further tuning was required. The creation of indexes – missing in BioPortal – followed.

Indexes are auxiliary performance-boosting data structures (more specifically, B-Trees in Microsoft SQL Server) that provide intelligent table lookup over a number of columns and ordered access to the desired information by avoiding the previous, meagre lookup method of having to read through the entire table, thus minimizing the amount of required I/O (the main bottleneck of any database). Indexes can be as relevant as complex as the databases' schema (and equally hard to correctly implement).

However, implementing indexes come at a cost: they require a large deal of storage and tend to slow data insertion and updating operation due to the fact that the indexes have to be rebuilt afterwards.

Indexes can be either clustered on non-clustered: clustered indexes occupy more disk space because the data is physically reordered, similarly to the way a dictionary orders its information, but provide an even greater performance booster when compared to non-clustered indexes. A significant downside of this kind of indexes is the fact that SQL Server has a maximum of one clustered index per table.

Non-clustered indexes, on the other hand, are similar to an index in the last pages of a book. The data is never physically reorganized; instead, pointers that show where the data is being stored are created. Since the data itself is not present in their B-Tree, they are also much smaller and slower than clustered indexes. Finally, each table can have up to 249 non-clustered indexes.

Indexes can also store data from two distinct fields at once: covered indexes. These indexes can provide fast access to data for queries that must lookup values in multiple columns at once. The

order by which these fields are ordered is also relevant towards the performance of the covered indexes.

Initially, the following indexes were created on the most commonly accessed fields of all tables (each bullet point is a distinct index inside a specific table):

Organism:

- TaxonomicId (Primary Key) – Unique, clustered index
- Organism Short Name and Organism "Long" Name – Unique, non-clustered covering index

Protein:

- ProteinId (Primary Key) – Unique, clustered index
- TaxonomicId– Non-unique, non-clustered index
- Locus – Non-unique, non -clustered covering index

Identifier:

- IdentifierId (Primary Key) – Unique, clustered index
- DataTypeId and Alias– Non-unique, non -clustered covering index
- ProteinId – Non-unique, non -clustered index

ProteinBioEntity:

- ProteinBioEntityId (Primary Key) – Unique, clustered index
- ProteinId – Non-unique, non-clustered index
- BioEntityId – Non-unique, non-clustered index
- DataTypeId – Non-unique, non-clustered index

BioEntity:

- BioEntityId (Primary Key) – Unique, clustered index
- DataTypeId and BioEntityName – Non-unique, non-clustered covering index

BioEntityDescriptiom:

- BioEntityDescriptionId (Primary Key) – Unique, clustered index
- Type and Name – Non-unique, non-clustered covering index
- Value – Non-unique, non-clustered index

DataType:

- DataTypeId (Primary Key) – Unique, clustered index

After implementing these indexes, a testing period followed. Although some queries, such as inner joining data from distinct tables, were now much faster, others clearly did not. Some of these cases – where indexes could never be used to retrieve the data – stand as expected; for example getting all gene names that end with '*a1*' where the index was useless due to the fact the indexes order strings from left to right. Since in this particular case only the last two letters mattered, having a column of strings ordered by their first letter would be pointless.

However, in other cases, better results were expected. For instance, little or no changes were noticed while obtaining all the existing gene names from the Identifier table, despite the presence of a non-fragmented, non-clustered covered index. It became apparent that adequately creating indexes is a complex issue and that the initial approach was clearly flawed; further tuning was required.

SQL Server has a built-in tool designed specifically for database optimisation: the DTA (*Database Tuning Advisor*). As such, this tool was chosen to aid in the development of proper indexes. In order to use the DTA, one of two approaches could be taken: either the tool analysed a set of pre-defined queries and suggested the corresponding database optimisations or the database and its requests could be monitored for a number of hours during a typical workload. After pondering both options, the former was chosen over the latter due to the fact that GeNS still has a rather small userbase and the results could have been tainted by this.

As such, five typical queries were chosen as benchmarks and executed three times in order to obtain more precise results:

1. All Gene Names
2. All Human Gene Names
3. All Human Protein
4. Convert HUGO Identifier HGNC:23600 the corresponding gene name
5. Search Pubmed entries for the human gene HLA-A

These workloads represent the most commonly performed operations for the GeNS databases, reaching almost every table and a very large number of concepts. The workloads were analysed by the Database Tuning Advisor, which suggested the creation of several covered indexes (on top of the already existing indexes) spawning multiple fields, as well as favoring some clustered

indexes on other fields, rather than on the unique identifier. In the Identifier table, for example, the DataType column was deemed appropriate for the creation of a clustered index; as such, the primary key clustered index was manually replaced with a non-clustered one.

The Tuning Advisor also proposed creating more statistics on multiple tables; these are vital in the determination of the optimal execution plan, thus making a decisive contribute regarding the fastest way to resolve a query. By default, SQL Server already keeps statistics over the data residing in individual columns but the Tuning Advisor can optimize the statistical generation process if, for a given workflow, it determines that additional statistics are required.

The following results were obtained:

Table 3.2 - Benchmarks

Query	Without Indexes (ms)			With Indexes (ms)			DTA's suggested indexes (ms)		
	#1	**#2**	**#3**	**#1**	**#2**	**#3**	**#1**	**#2**	**#3**
All Gene Names	26949	27198	27248	27174	26862	26698	17188	17560	16986
All Human Gene Names	27698	28447	27942	24971	25359	25311	1869	1856	1743
All Human Proteins	2401	2405	2523	4558	4525	4548	885	779	778
Convert Identifier	66943	67254	67167	52849	53320	52825	166	187	190
Search Pubmed	46012	46802	46014	7505	7422	6896	2756	2716	2573

Table 3.3 - Average query response time and improvements over the non-indexed version

Query	Average (ms)			Improvement (%)	
	No Indexes	With Indexes	DTA's indexes	With Indexes	DTA's indexes
All Gene Names	27132	26911	17245	1%	57%
All Human Gene Names	28029	25214	1823	11%	1438%
All Human Proteins	2443	4544	814	**-46%**	200%
Convert Identifier	67121	52998	181	27%	**36984%**
Search Pubmed	46276	7274	2682	536%	1626%

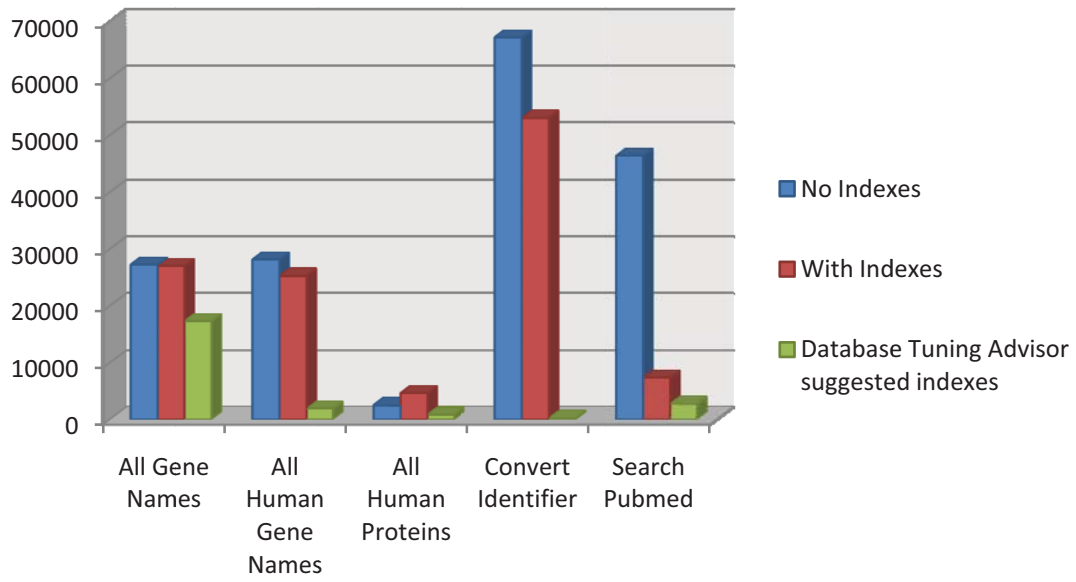

Figure 3.11 – A chart comparing the three approaches (in milliseconds)

As one can clearly see, the new indexes suggested by the Database Tuning Advisor had profound effect on the system's performance. The overal response time of all queries was greatly improved and, in some cases, reduced to a fraction of the original. The Convert Identifier query, one of the pillars of GeNS' features, is now 370 times faster, for instance.

There was, however, one result that cannot be easily explained – the loss of performance in the query that selects all human proteins regarding the initial table index implementation (-46%). After investigating for quite some time no clear answer was found; a possible explanation is that SQL Server is incorrectly choosing the best way to access a table (which can either be direct or indirect, whether the data is accessed directly or indirectly by reading an index and fetching the data accordingly).

Having a large number of indexes may be causing SQL Server to pick the slowest way of retrieving the data. This problem generally comes up when either the statistics regarding a particular table in the database are incorrect, when the indexes are fragmented or when large sets of data were introduced in the table. During this experiment, the indexes were only created after having generated a fresh set of statistics; it is possible that the statistics required a longer period of time in order to improve the system's performance. Therefore, it is entirely possible that running a non-indexed version is faster than a poorly implemented indexed version, albeit uncommon.

Either way, an alternative solution for this issue would be to force SQL Server to ignore the indexes and to perform a table scan (as if no index existed) by means of hints. Using hints is not recommended because SQL Server is usually very efficient in determining the optimal way of accessing the table rows. Nevertheless, in this particular case, a query hint would likely solve the issue. To do so, the SQL query would have to be slightly modified to the following (where bold indicates the changes):

*SELECT * from Protein* ***WITH (INDEX(0))*** *where TaxonomicId = 9606*

Adding it all up, the Database Tuning Advisor played a crucial role towards GeNS performance. As expected, however, this huge performance boost came at a price: disk space. The DTA led to the creation of six gigabytes of indexed data, roughly a quarter of the size of the database. Nevertheless, the advantages are so big that the added disk space requirements are nothing more than a nuisance.

The full list of indexes – where the suggested indexes are underlined – follows:

Organism:

- TaxonomicId (Primary Key) – Unique, clustered index
- Organism Short Name and Organism "Long" Name – Non-unique, non-clustered covering index

Protein:

- ProteinId (Primary Key) – Unique, clustered index
- TaxonomicId – Non-unique, non-clustered index
- Locus – Non-unique, non-clustered covering index

Identifier:

- <u>IdentifierId (Primary Key) –Unique, non-clustered index</u>
- <u>DataTypeId – Non-unique, clustered index</u>
- <u>Alias, DataTypeId and ProteinId – Non-unique, non-clustered covering index</u>
- <u>DataTypeId and ProteinId – Non-unique, non-clustered covering index</u>
- <u>DataTypeId and Alias – Non-unique, non-clustered covering index</u>

ProteinBioEntity:

- <u>ProteinId and BioEntityId – Non-unique, non-clustered covering index</u>

- DataTypeId – Non-unique, non-clustered index

BioEntity:

- BioEntityId (Primary Key) – Unique, clustered index
- DataTypeId and BioEntityName – Non-unique, non-clustered covering index

BioEntityDescriptiom:

- BioEntityDescriptionId (Primary Key) – Unique, clustered index
- Type and Name – Non-unique, non-clustered covering index
- Value – Non-unique, non-clustered index

DataType:

- DataTypeId (Primary Key) – Unique, clustered index

Concluding, creating adequate indexes is a complex task that requires a lot of adjustments and considerations. Even with the help of the DTA, there is still room for future improvements. The increasing userbase may prove fundamental in determining what can and should be optimized during a typical workload.

Finally, an additional measure could have been taken towards increasing GeNS' performance: table partitioning [65]. Table partitioning consists in splitting data from a table into smaller sections, thus improving the table's scalability due to the fact that the DBMS no longer has to transverse through the entire table or through an index in order to access the desired records. Instead, the DMBS can jump to a specific section and read a much smaller amount of data. On top of this, indexes in each section can be created in order to further enhance the system's performance.

GeNS has not implemented table partitioning because the indexes have bestowed the system with very good query response times, skipping the need to do so entirely. This technique can, however, be easily implemented in the future; the DTA, for example, has a built-in support for this kind of analysis that may shed some light on the matter.

3.5.7. Maintenance and data recovery

It is necessary to apply maintenance operations on the database if the system is to work properly in the long run. Due to their repetitive nature, these tasks can and should be automatically

processed, especially since SQL Server already supports them via SQL Server Agent's jobs and maintenance plans.

Two SQL maintenance scripts were implemented in order to keep the database running smoothly over long periods of time. The first script runs on daily basis and purges the database of any temporary tables that may linger, while the latter one is ran weekly during low-usage hours (starting every Saturday at midnight) and performs several important tasks such as checking the database integrity, rebuilding indexes and updating statistics (Figure 3.12).

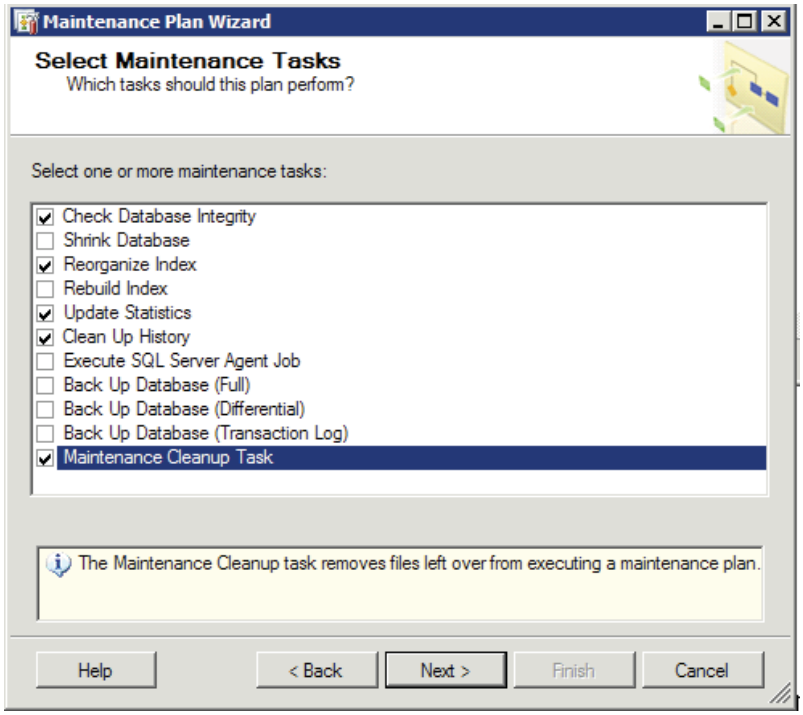

Figure 3.12 - Creating GeNS' maintenance task in SQL Server

In addition, a compressed backup of the database is formed every Sunday at midnight on two distinct physical drives: the slave hard disk and an external hard disk (if connected to the computer). Compressing a backup of a database this large is a CPU-intensive task that must be controlled in order to ensure the system's usability during the procedure. To do so, a new account with severe CPU restrictions (using a maximum of twenty percent of the CPU's capacity) was created and implemented via SQL Server's Resource Governor. This implementation allows the users to keep using the database, despite of the complex task being executed in background.

Finally, all of these scheduled tasks send an email notifying the database administrator of the outcome of their latest execution.

3.5.8. Flaws

This tier has three problems: updating large sets of data, integrating complex biological entities' descriptions and mapping relationships between biological entities.

Regarding the first, updating the data sets is the most relevant problem in this tier. The main issue is the sheer size of the dataset and the transformations the data suffers before its actual integration in the database; because of these two factors, comparing two sets of data becomes a very CPU intensive task. As such, although a valid option for smaller data sets (e.g. KEGG Drug), this method is not feasible for considerably larger data sets such as UniProt's.

Therefore, the only currently available way of updating a large data set is to simply delete the older data and to integrate the new one instead. This process can cause serious problems to applications running on top of GeNS that have static references to the entries in the database, since both internal identifiers and the objects themselves are subject to change with the update.

A possible solution for this issue is to perform a differential update by comparing the last modified date in UniProt's source files with the one inside GeNS' database, thus performing a selective update instead. This is, however, a theoretical solution because no current implementation of this method is in practice.

As for the second issue, certain kinds of descriptions are simply too complex for the current implementation of the BioEntityDescription table. Practically speaking, only three fields are available to store this kind of data – *Type*, *Name* and *Value* – and some descriptions may require additional fields, such as ArrayExpress Atlas' gene expression data; in this particular case, the data has to be concatenated in order to fit the available fields. A greater degree of data normalization is required in order to provide a better way of mapping these descriptions, quite possibly envolving the DataType table as well.

Finally, relationships linking one biological entity to another are not being integrated due to the lack of support in the physical schema. These relationships are important because a number of biological entities are directly related with other ones, such as the pathways involved in a particular disease, for example. Being able to map these relationships directly would allow GeNS' users to quickly gain a detailed overview on a particular matter without using complex queries. This could be achieved by either adding an extra field in this table that referenced other biological entities or by introducing a correlation table.

While the former option is presents a rather simple way of mapping a one to one relation between two biological entities, it cannot be used in this case because of the multiple to multiple nature of the relationships.

Adding a correlation table, on the other hand, not only solves the problem but it also accounts for a more flexible, consistent and easy to maintain implementation that follows good normalization practices.

3.6. Data Presentation Tier

The data presentation tier is responsible for making the integrated data available to any user who so desires. To do so, two distinct access methods are available: direct SQL access and a Web Services interface. While the former enables third-party applications to access the database, the latter allows users to access GeNS' data without having installed a local instance of the platform.

GeNS' Web Services can be found at http://bioinformatics.ua.pt/GeNS/WS/ and can be used by any Web Browser or accessed directly by programs developed in any language that supports SOAP/WSDL. While supported by Microsoft's Internet Information Services (IIS), these Web Services return the data in XML format. Online documentation is also available in GeNS' website. The following methods are available to use:

- ✓ List Data Types
- ✓ Search Protein
- ✓ SearchProteinStartsWith
- ✓ SearchProteinMatches
- ✓ Search Biological Entity
- ✓ Search Organism
- ✓ Convert Protein Identifier

However, there is more than meets the eye: each of these methods accepts a large number of parameters and, according to their input, performs a different operation. This implementation makes the Web Services layer easier to use, although inexperienced users may find it confusing because of the hidden functionalities. As such, GeNS' web site at http://bioinformatics.ua.pt/applications/gens contains an online documentation section that gives an insight on the flexibility of these Web Services methods.

Table 3.4 - The ListDataType method

Method	Results
ListDataType()	Returns all known datatypes.
ListDataType(search_name_string)	Returns all known data types matching the search_name_string variable
ListDataType("Identifier")	Returns all data types present in the Identifier table
ListDataType("BioEntity")	Returns all data types present in the BioEntity table

As the name points out, these methods are used to list GeNS' data types (used to indicate the nature of the data itself for protein identifiers and biological entities). The two last methods enable GeNS' users to quickly determine what kind of data is being stored in the Identifier and BioEntity tables.

Table 3.5 - The SearchOrganism method

Method	Results
SearchOrganism():	Returns the first one thousand organisms that have both scientific and short name in the database
SearchOrganism(search_name):	Returns all organisms that have either a scientific or a short name matching the search_name variable
SearchOrganism(search_taxonomicid):	Returns all organisms that have a matching taxonomical Id with a matching search_name variable

These methods allow the users to search for an organism's scientific name, short name or taxonomical id, providing organism related data.The first method provides an insight regarding the organisms integrated in the GeNS platform (which range 180.000 entries).

Table 3.6 - The SearchProtein method

Method	Results
SearchProtein():	Returns the names of the first one thousand proteins of the Homo Sapiens organism
SearchProtein(taxonomic_id, datatype):	Returns data of the specified data type of the first one thousand proteins of the organism that has a matching taxonomic id.
SearchProtein(taxonomic_id, datatype, limit):	Returns data of the specified data of the first X proteins (where X is defined by the limit parameter) of the organism that has a matching taxonomic id.
SearchProtein(taxonomic_id, datatype, lower_limit, upper_limit):	Returns data of the specified data type of all proteins between the lower and top limit (defined by the last two

	parameters) of the organism that has a matching taxonomic id.
SearchProteinStartsWith(alias)	Returns the first one thousand (proteinic) alias that match the user-inputted text. In practical terms, it's an SQL wildcard search that will look for *<alias value>**
SearchProteinMatches (alias)	Similar to the SearchProteinStartsWith method, but the user-inputted text will be searched everywhere inside the alias. In practical terms, it's an SQL wildcard search that will look for **<alias value>**. This method may take some time.
ConvertIdentifier(taxonomic_id, alias, datatype)	Returns all identifiers of the specified data type for any protein (belonging to the organism with a matching taxonomic id) that has a matching alias

The last method - Convertdentifier - is one the main features of GeNS; being able to convert an identifier from one data type to another, yet related to the same protein, is a very important attribute as it enables the creation of a unified view of data coming from distinct sources.

In a typical scenario, a user - Bob - is trying to determine the HUGO Gene identifier for the human protein *KRTAP5-6*, for instance. After determining the Homo Sapiens' taxonomic id – 9606 – (by accessing the Organism table) and looking up HGNC's datatype – 34 – (through the DataType table) Bob uses his Web Browser to execute: ConvertIdentifier(9606, KRTAP5-6, 34)

GeNS returns:

− *<ArrayOfProteinIdentifier>*

− *<ProteinIdentifier>*

<Alias>HGNC:23600</Alias>

<DataTypeId>34</DataTypeId>

</ProteinIdentifier>

</ArrayOfProteinIdentifier>

As we can see, the human protein KRTAP5-6 has an HGNC identifier: HGNC:23600.

Table 3.7 - The SearchBioEntity method

Method	Results
SearchBioEntity():	Returns the names of the first one thousand biological entities belonging to the *Homo Sapiens* organism
SearchBioEntity(bioentityname);	Returns all the biological entities matching the user-inputted BioEntityName.
SearchBioEntity(search_taxonomicid, datatype):	Returns data of the specified data type for all proteins belonging to an organism with a matching taxonomic id.
SearchBioEntity (search_taxonomicid, alias, datatype):	Returns data of the specified data type for any protein (belonging to the organism with a matching taxonomic id) that has a matching alias.

Each method's input parameters are validated and, according to their number and kind, the appropriate sub-method will be selected and performed. Each sub-method creates an SQL connection (if none exists) to the local database and executes the desired query. Some of these sub-methods also require multiple sub queries and the creation of temporary tables in order to enhance their performance.

Finally, the data is stored into DataView instances and used to create a statically defined type of data which, in turn, will be returned in XML format to the user.

Security

While the implementation of these Web Services greatly improved GeNS' accessibility, it also opened a doorway for malicious users who may wish to tamper with the database. Initially (and not surprisingly), the Web Services were vulnerable to all kinds of attacks (e.g. SQL injection [66]). For example, if a malicious user were to execute the ListDataType method with the following input:

'; DROP DATABASE GeNS --

Then SQL Server would effectively drop the entire database thus rendering the entire system unusable and requiring manual intervention to restore the database (assuming a backup copy existed); this is an unacceptable risk for the platform.

However, due to the fact that efficient application hardening is a rigorous, time-consuming task that is out of the scope of this thesis, a set of simple measures were implemented in order to minimize the issue:

- Perform basic server-side input validation
- Prevent the execution of arbitrary code by restricting the Web Services SQL commands to parameterized Stored Procedures, executed by a user with limited permissions
- Disabling the guest account inside SQL Server 2008

Regarding the first, validating the inputted text on the server prevents malicious users from circumventing the client-side validation process, albeit at the cost of additional workload for the server; despite effectively shutting down the former attack, this decision may pave the way for a Denial of Service attack if a skilled, malicious user so desired. Even so, it raises the difficulty bar and, as such, it is still preferable to client-side validation since only validated parameters are passed along to SQL Server's stored procedures.

About the validation process itself, upon receiving a request from a Web Service that accepts a string in its input parameters (the gateway for most SQL injection attacks). Subsequently, a validation method is invoked in order to sanitize the user-inputted string.

More specifically, a number of characters and substrings typically used in SQL injection attacks were blacklisted; these will be removed from the input string, if found, thus effectively stopping the SQL Injection attacks. Amidst the blacklisted items one can find the single quote character ('), the comment operators (--), and a number of key SQL commands (e.g. DROP, DELETE, SELECT, INSERT, BACKUP, xp_cmdshell, TAKE CONTROL). All the user-inputted integers must also be validated.

Regarding the second measure, a user with a very restricted set of permissions was created in order to further restrict malicious intents; all operations are executed by means of parameterized stored procedures, following recommended practices [67] and conferring a second layer of protection to the system.

Finally, disabling the guest account may prove useful if the attacker manages to overcome the remaining protections; a guest account would allow anyone to connect to the database, an irrelevant feature for the time being.

These measures appeared to solve the problem regarding the most commonly used SQL Injection attacks. Nevertheless, one can never be too careful and performing penetration testing may prove quite useful in the future as this would certainly put the implemented measures to the test (if lead by an experienced user, obviously).

3.7. Summary

In this chapter, the GeNS platform was presented as a multi-tier, database-centric data integration system that is able to combine features from link-based, mediators and, especially, data warehousing strategies to integrate large sets of data, composed of numerous biological concepts along with their relationships.

Several loaders were developed in order to provide access to multiple sources, along with SQL scripts that import the data to the DB.

The physical database schema improved BioPortal's schema regarding data replication (minimized by the new protein-to-biological entity correlation table), intelligibility and performance. In addition, indexes and statistics were created in order to boost performance even further. Maintenance scripts were developed in order to keep the database up and running for long periods of time.

The data was made accessible by direct SQL access and a SOAP based Web Services layer that provides multiple functionalities and whose data is returned in XML format.

Chapter 4 – Validation procedures

This chapter describes the validation procedures and the concrete applications using the GeNS platform. These are particularly important, because a system such as this one cannot be correct in itself; it requires a working application working on top of it in order to judge if the concrete level of abstraction, for example, is suitable for the task at hand.

4.1. Programmatic utilization

By June, 2009, GeNS was being used in two distinct applications: QuExT [68] and GeneBrowser [69].

QuExT (*Query Expansion Tool)* is a web application designed to search biomedical literature in order to find relationships among sets of genes. For a given list of genes, it expands the initial search in several biological domains using a mesh of co-related terms, extracts the most relevant document from the literature, and organizes them according to domain weighted factors. The role of GeNS database is to retrieve the network of concepts related with each gene entry in order to perform the query expansion.

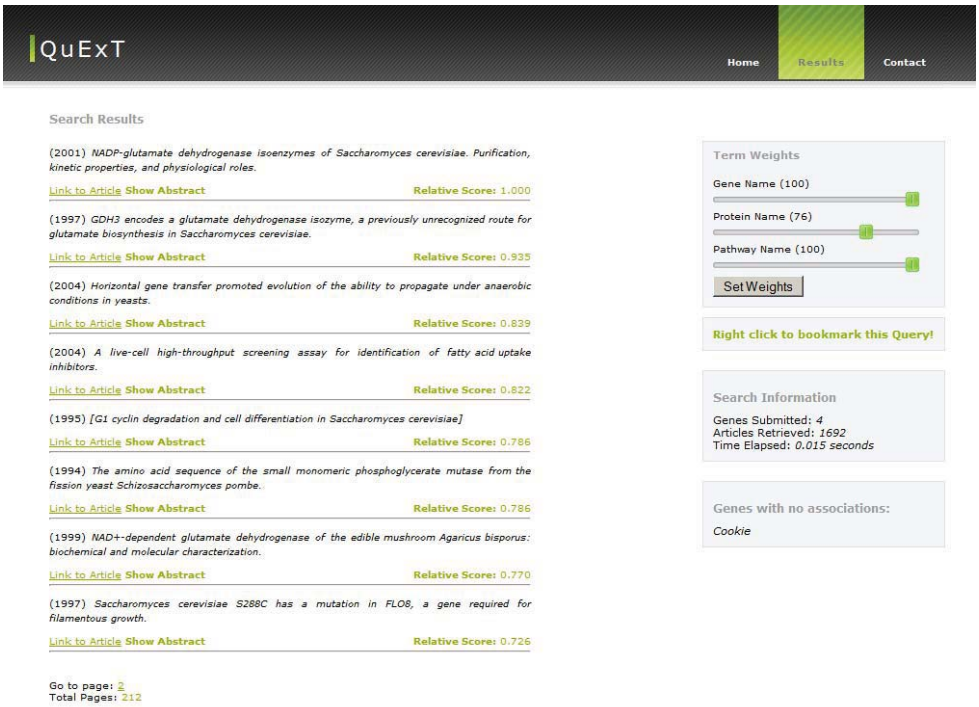

Figure 4.1 - QuExT's search results for its example query

GeneBrowser, on the other hand, is a web-based application that offers to the user several interpretation perspectives to help giving biological significance to the result coming from a DNA-microarray experiment.

For a given set of genes the system obtains and shows to the user relevant information extracted from external databases. Other features of the system include the possibility to see the accumulation of genes into several categories (Pathways, Gene Ontology terms and KEGG Orthology terms).

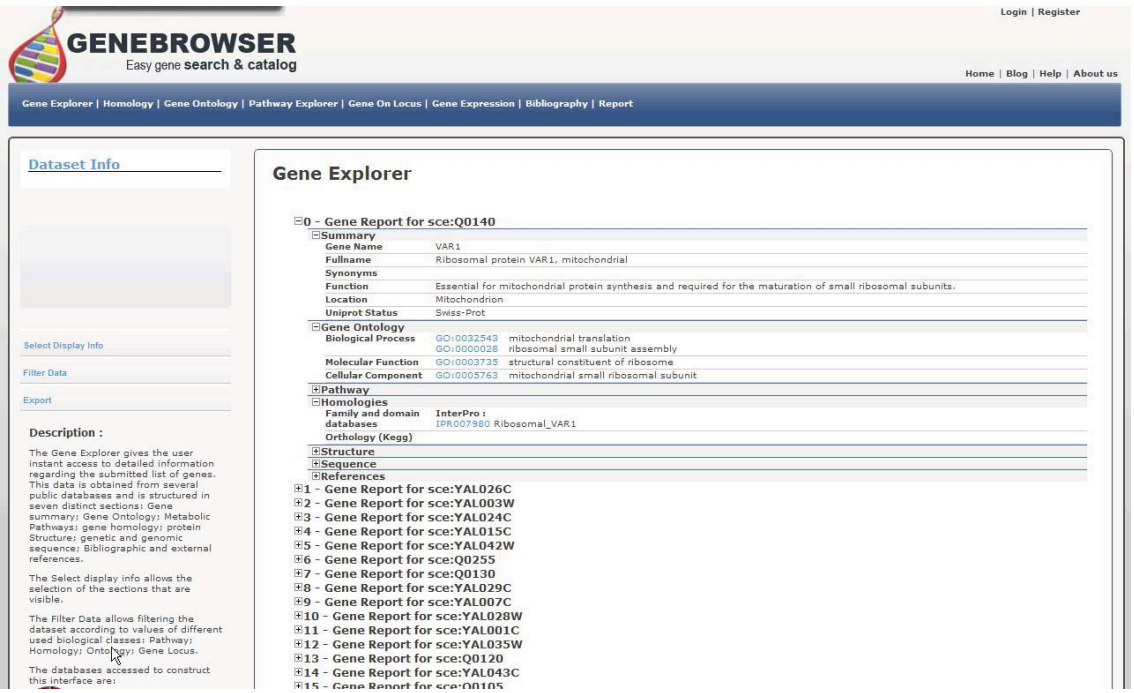

Figure 4.2 - GeneBrowser's Gene Explorer in action

These tools are a test to GeNS' accuracy; both were able to audit the results provided by the GeNS platform, confirming the correctness of the inserted data.

4.2. Comparing with DiseaseCard

DiseaseCard is *"an information retrieval tool for accessing and integrating genetic and medical information for health applications"* developed in the UA.PT Bioinformatics Group in collaboration with the Institute of Health Carlos III Bio-Computing and Public Health Unit [33].

DiseaseCard receives data exclusively from NCBI OMIM and Clinical Trials databases via the Arabella crawler and compiles all the data into disease cards that group all the data regarding a given disease into a single entry point. This is a mature, well established system and, as such, a

side by side comparison would provide a clear indication of GeNS' correctness regarding data integration (for the human species only, as DiseaseCard only integrates *Homo Sapiens* data only). For this purpose, five diseases were selected:

- Alzheimer's disease
- Renal tubular dysgenesis
- Glioblastoma
- Gastric cancer
- Osteosarcoma

All of these diseases have complete cards in DiseaseCard (20 out of 20 informational nodes, the maximum amount of information available); this is important because if a disease with an incomplete set of data was selected then the results may not have been accurate, especially since DiseaseCard has a low number of sources (unlike GeNS).

Having selected the diseases, their OMIM number was looked up and ran through DiseaseCard and GeNS. A small software application (built in C#) was also developed in order to deliver automatic processing of the results.

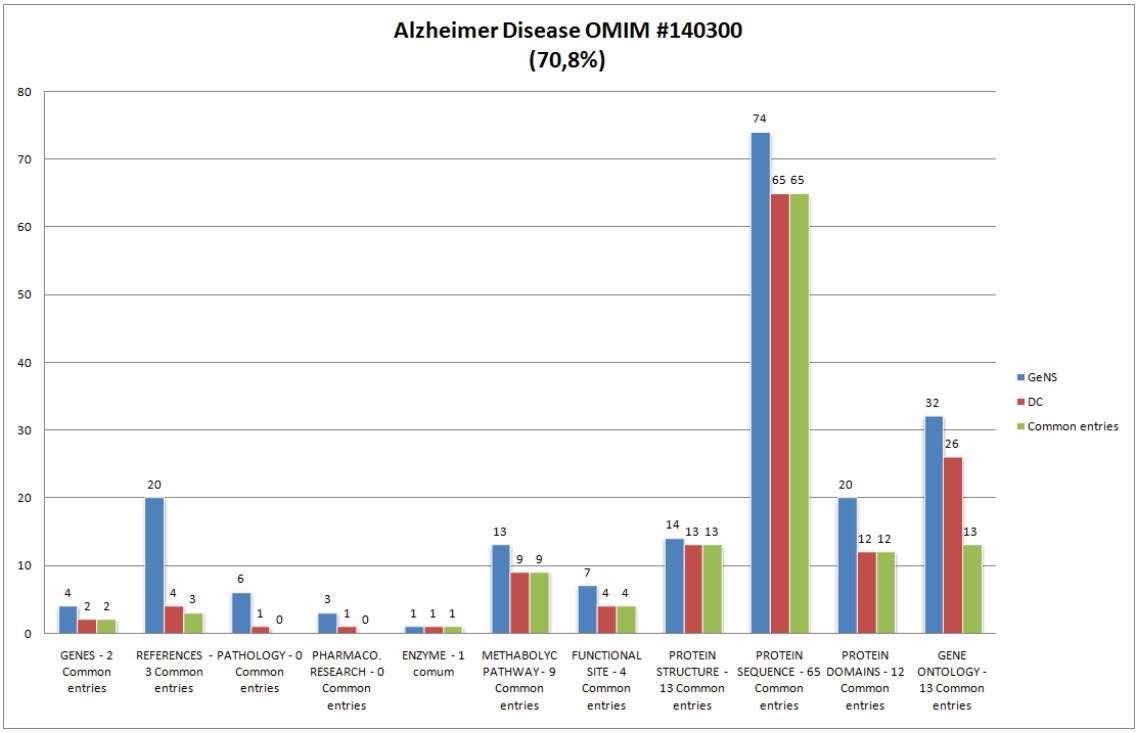

Figure 4.3 - A comparative analysis for Alzheimer's Disease

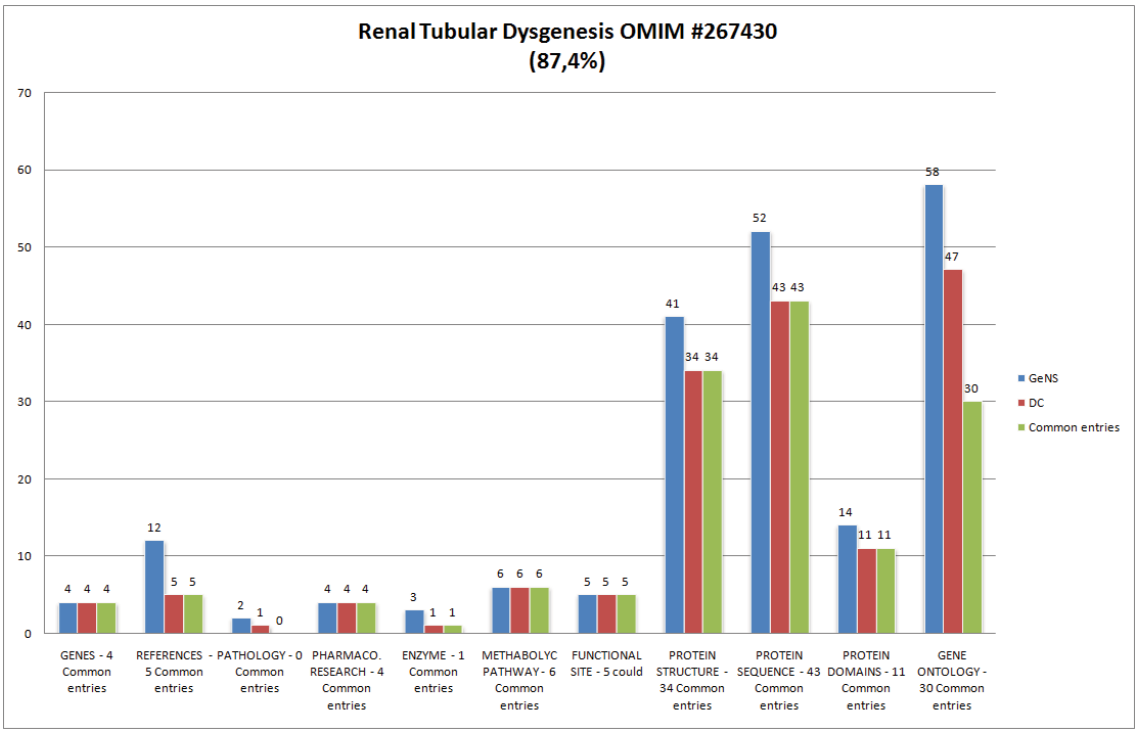

Figure 4.4 - A comparative analysis for Renal Tubular Dysgenesis

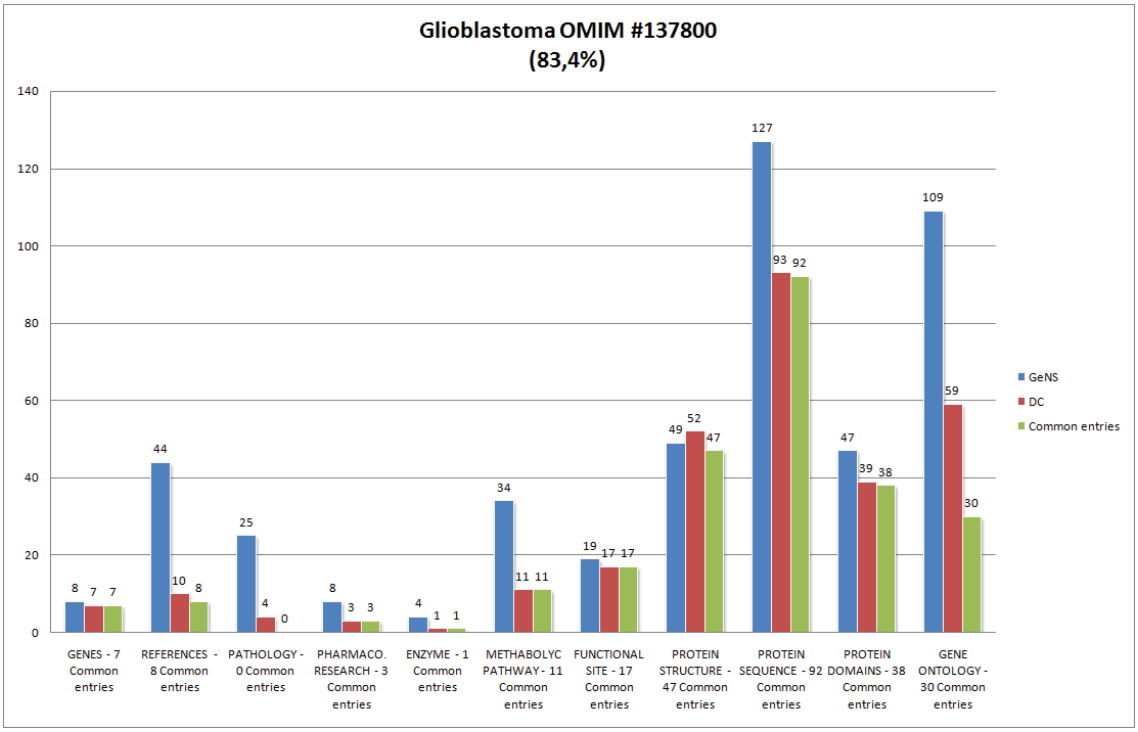

Figure 4.5 - A comparative analysis for Glioblastoma

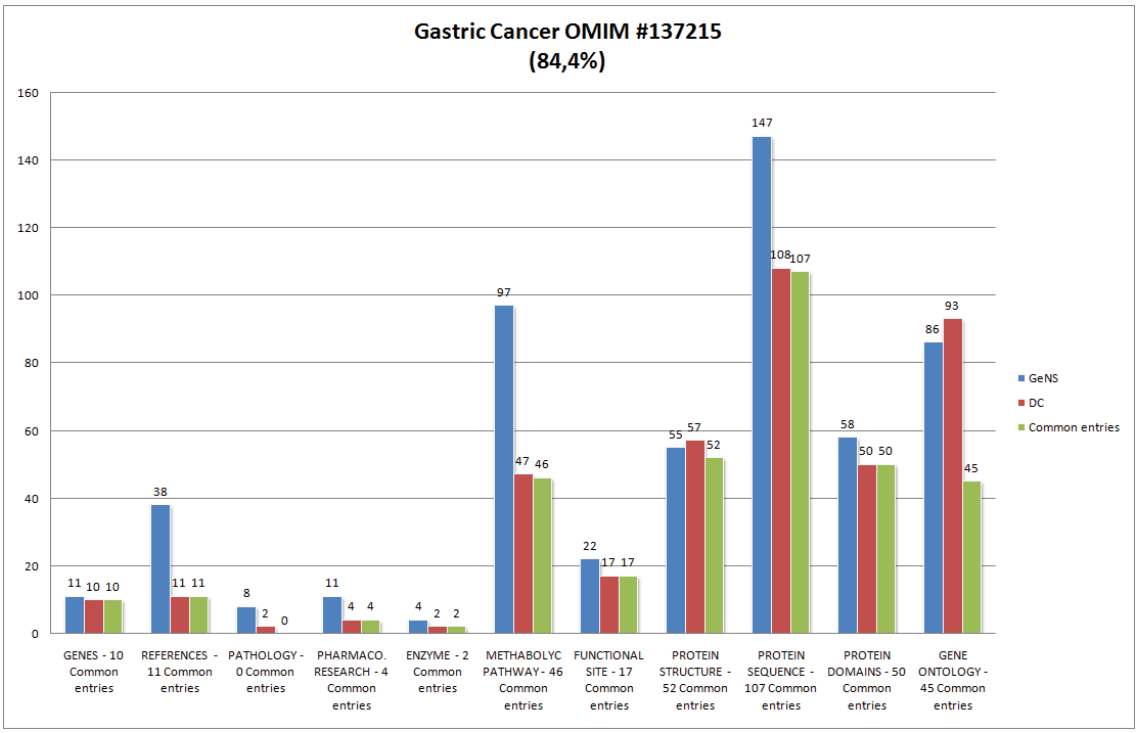

Figure 4.6 - A comparative analysis for Gastric Cancer

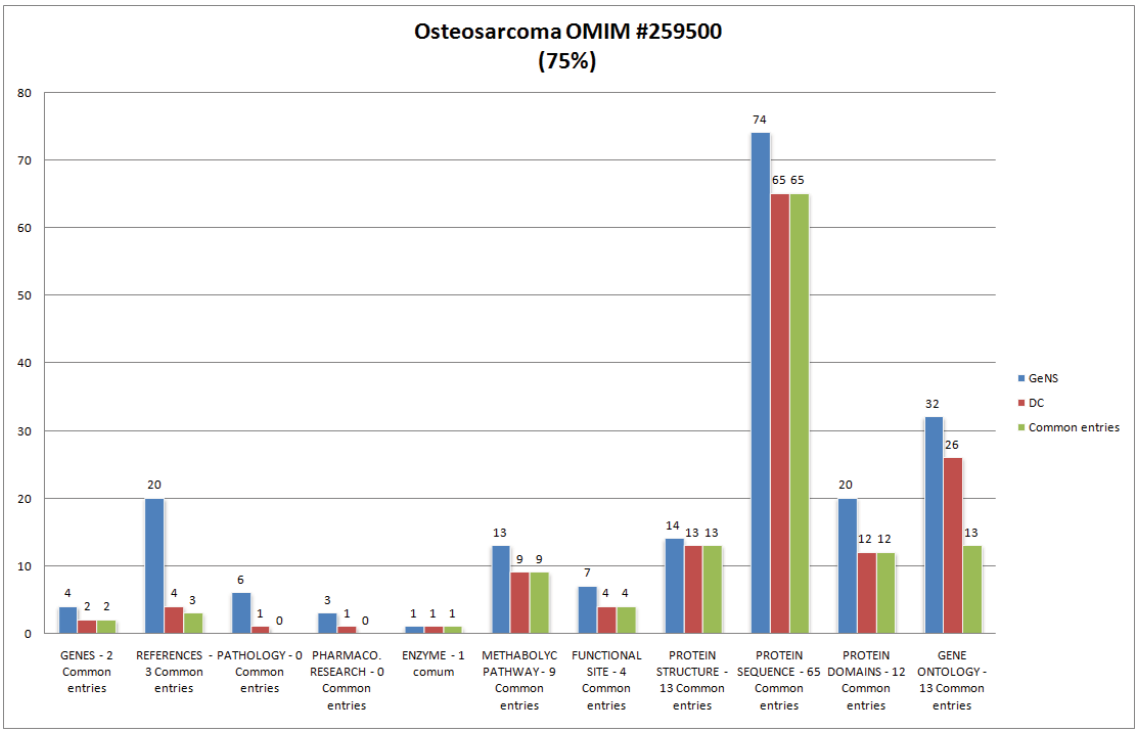

Figure 4.7 - A comparative analysis for Osteosarcoma

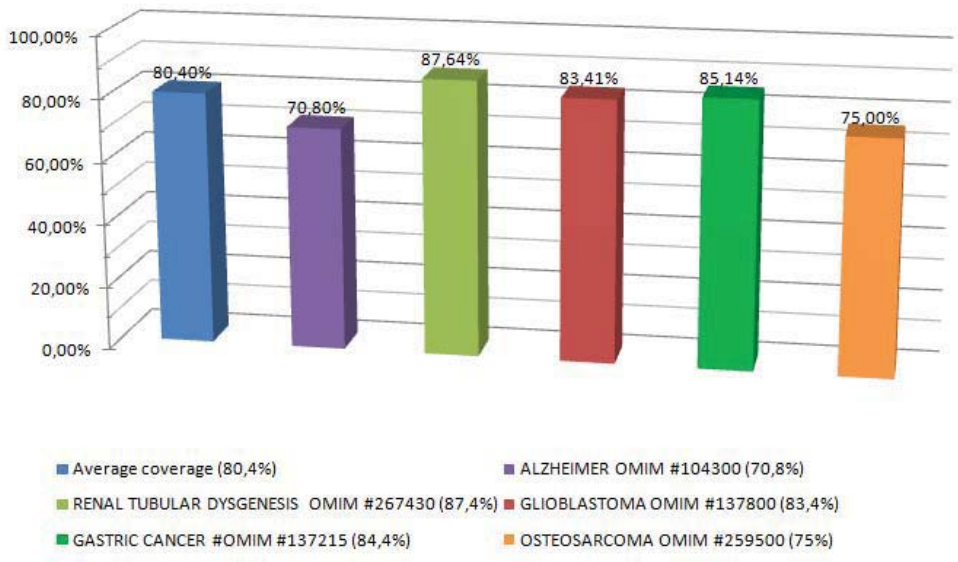

Figure 4.8 – The coverage rate between GeNS and DiseaseCard

As the graphics clearly demonstrate, the GeNS platform is able to provide a good coverage of the data already in DiseaseCard (in terms of common entries), while presenting even more results in nearly all of the queries - an expected result due to the large number of sources feeding its database. Although theoretically GeNS and DiseaseCard should have an a hundred percent coverage rate, these databases integrate information with different timestamps. As a consequence, this rate falls to an average coverage of eighty percent (Figure 4.8).

Therefore, one can conclude that GeNS is correctly integrating human-related disease data. Also visible is the importance of a periodic update for its database, as the coverage rate between both systems will eventually decline if GeNS does not update its data regularly.

4.3. Comparing with Bio2RDF

Bio2RDF is a semantic web mashup system developed at Laval University that integrates biological knowledge from different data sources (GeneID, OMIM, UniProt, KEGG, Ligand, OBO, PDB and MGI, among others) whose goal is “to solve the problem of knowledge integration in biology by applying a semantic web approach” [70]. It is this semantic web approach that makes Bio2RDF an interesting choice in comparison to GeNS.

Bio2RDF converts and stores data in RDF (Resource Description Framework) data model [71] format in order to implement the semantic web technology. This W3C recommended standard model is flexible enough to support conceptual description or modeling of information from

different data sources. A morphological equivalent to a collection of RDF statements would be a labeled multi-graph that clearly indicates the flexibility of this approach. As such, implementing a web semantics layer bestows any system with complex query searches, along with full text search, due to the large amount of metadata associated with a given biological concept; this is one of the pillars of Bio2RDF.

Architecture

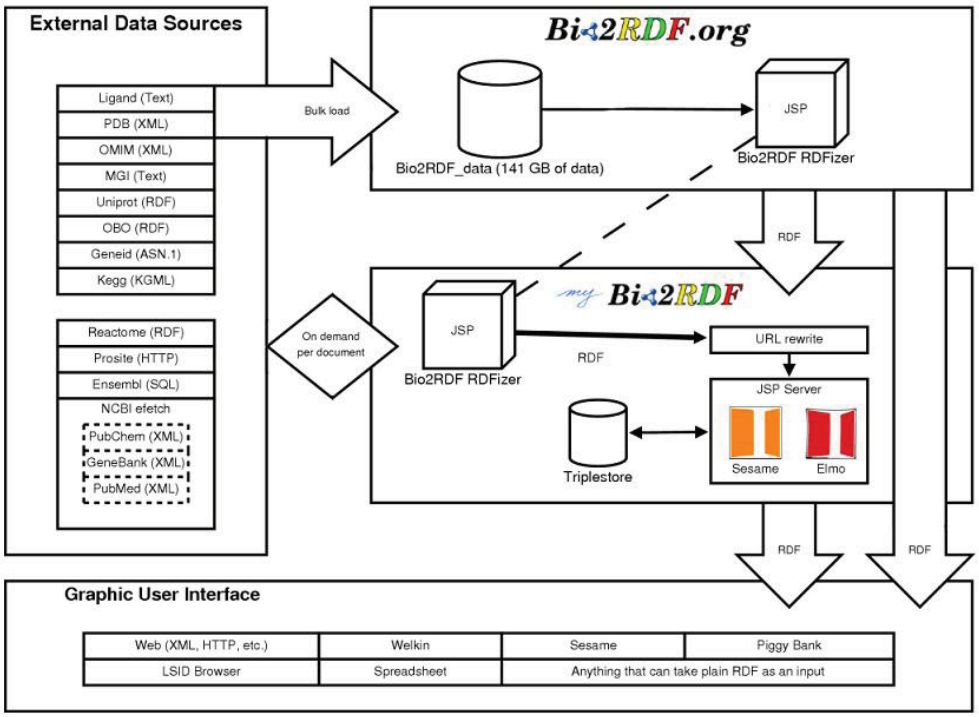

Figure 4.9 - Bio2RDF's architecture

Briefly summarizing its complex architecture, Bio2RDF integrates data from a number of sources (PDB, OMIM, UniProt, KEGG and Ligand, among others) in RDF format only. As such, data being retrieved not in RDF format needs to be converted to it via an *Rdfizer* program developed using the JSP [6] toolbox. The data is subsequently stored in a local MySQL database and made available via several graphic user interfaces.

This represents the data warehouse facet of this system, used to store the metadata that will not only show some results but also provide an effective inference method of retrieving new data from other third-party databases on a per request basis, the mediator facet of Bio2RDF.

[6] Available at http://java.sun.com/products/jsp/jstl/

This greatly increases Bio2RDF's namespace, even though its performance drops due to the obvious need to contact other sources of information. Nevertheless, the resulting data can be subsequently integrated in the database in order to avoid waiting long periods of time in the future.

Bio2RDF supports very powerful queries that enable answering both technical and abstract question such as[7]:

- Who is Jean Morissette?
- Who are Dr Labrie collaborators?
- Which MeSH terms are associated to Dr Labrie papers?

These queries prove Bio2RDf's powerful querying capacities; as it is, they cannot be answered by GeNS. As one can see, Bio2RDF follows a different approach than the one being used in GeNS. Its RDF based implementation is inherently more complex in nearly all aspects (Figure 4.9 taken from [70]).

Developers wanting implement applications that run on top of it either use one of the existing services or acess the database directly via SPARQL [72]. Creating SPARQL queries for unexperienced developers in this area may prove cumbersome at the beginning due to the low intelligibility of the language itself; for example, Bio2RDF's script for retrieving all the genes involved in the KEGG Pathway 'path:mmu0010' is:

```
SELECT distinct ?label1, ?sameAs5, ?xobject4
WHERE {
        ?Pathway1 <http://www.w3.org/2000/01/rdf-schema#label> ?label1 .
        ?Pathway1 <http://bio2rdf.org/kegg#xrelation> ?xrelation2 .
        ?xrelation2 <http://bio2rdf.org/kegg#xentry1> ?xentry3 .
        ?xentry3 <http://bio2rdf.org/kegg#xobject> ?xobject4 .
        ?xobject4 <http://www.w3.org/2002/07/owl#sameAs> ?sameAs5 .

FILTER (?Pathway1 = <http://bio2rdf.org/path:mmu00010>)
}
```

[7] Taken from http://bio2rdf.wiki.sourceforge.net/Demo+queries

As a result, developing applications on top of this system may take considerable longer when compared to the same task while using GeNS. Additionally, modifying a query to deal with recently added types of data should not pose problems in either GeNS or Bio2RDF.

The results are typically published in RDF format, even when manually querying the system via its main website (http://www.bio2rdf.org/), a fact that may cause some "discomfort" to unexperienced users as RDF is, nevertheless, still rather unknown to the general public and, albeit human-recognizable, it requires some effort to be read. This issue was minimized with the implementation of a number of alternative interfaces that simplify the system's usage.

Both systems have a similar amount of conceptually different types of data, mostly due to the also similar number of third-party sources. Nevertheless, Bio2RDF has much stricter disk requirements (occupying 141 GB of disk space in 2007, according to [70]). This is direct consequence of the RDF approach; having to store such a large metadata, along with the verbosity of the RDF format itself translates into a massive volume of data. GeNS, on the other hand, requires 26GB for storing data about 180.000 species – the main differente is that a large percentage of the data is comprised of identifiers and no metadata (to the obvious exception of the data type identifier that specifies the nature of a given entry in the database) is stored.

4.4. Summary

All things considered, the GeNS platform provides a solid, coherent foundation for a wide range of applications; it returns a vast amount of scientifically correct data (as proved with the DiseaseCard comparison) in a short response time (as proved with the GeneBrowser application).

GeNS and Bio2RDF use distinct strategies towards the same goal; while the power of the semantic web technology makes Bio2RDF rather interesting, its limited set of data regarding species, the heavy disk space requirements and its complexity may drive away potential users. On the other hand, GeNS only supports "semanticless" queries that cannot offer the same raw searching power; as such, implementing even a simple semantic layer on top of the database may be able to fill this gap and provide interesting options for this platform.

Chapter 5 – Conclusions and future work

The integration of heterogeneous data sources is a classic problem in Bioinformatics, where the ability to provide a unified view of conceptually different sets of data offers scientists a much broader view of a given subject, thus making it much easier to extract conclusions that may not have been visible otherwise.

In this thesis, the GeNS platform is introduced as an alternative solution. Having identified the main requirements, potential data sources and the characteristics of a large number of third-party solutions, a detailed architecture was proposed, implemented and polished. The main contributions of this tool are its easiness to use and maintain while offering great performance. Moreover, its coverage and scalability – both in terms of number of sources and types of data – can easily be verified by the sheer number of distinct types of data already integrated. Adding new ones is a trivial procedure and, in all odds, new kinds of data will be incorporated in a nearby future. The large coverage also enables GeNS to act as a name server by converting an identifier to one of a different nature but associated with the same object; this feature is directly available via Web Services.

The proposed methodology to improve the coverage of the database will likely play a vital role in the nearby future in order to ensure the completeness of the stored data. This claim is corroborated by the promising experimental results that clearly show that substantial improvements were made in almost all database relationships. No third-party solution has implemented an equivalent methodology, a fact that increases its relevance even more. As such, more work should be placed in this vertent in order to improve the platform even further.

The current instance of GeNS integrates some of the most relevant molecular biology databases available and nearly 180 million entries in the local database, occupying 26 GB of disk space. To show its functionality, two distinct applications – GeneBrowser and QuExT - are supported by GeNS services.

The first one is a tool that performs the functional analysis of microarray data and the second uses GeNS data to improve text mining results over PubMed.

All things considered, it becomes clear that nearly all proposed objectives were achieved, the only exception being the lack of an efficient update system. However, this operation is theoretically viable because the largest set of data (UniProt) always has a modification time

associated with every entry, which can be used to verify if the data regarding a given protein has been updated since it its last integration in the local database.

All in all, GeNS represents a major improvement regarding its predecessor and has a set of features that allows it to compete head-to-head with many third-party data integration solutions.

5.1. Developed skills

A lot was learned during the development of the GeNS platform, starting with a deeper knowledge of C#, XML, ASP.NET and SOAP-based Web Services, as well as effective database design, maintenance task and database optimisation.

Microsoft's Visual Studio and SQL Server Management Studio in particular were two crucial tools for GeNS' successful deployment; the latter one required further exploration in order to gain benefit of several of its features (e.g. Resource Governor, DTA).

Substancial knowledge was also drawn from the operating system running GeNS: Microsoft's Windows Server 2007 Enterprise, along with the IIS web server that supports the Web Services layer.

Finally, the collaborative group work performed within the Bioinformatics group was a rewarding experience that sharpened my social skills and effectively provided a solid basis for corporate life.

5.2. A SWOT Analysis

The following SWOT analysis evaluates all the aspects of the developed work by identifying its **s**trengths, **w**eaknesses, **o**pportunities and **t**hreats. This facilitates future work by making it easy to identify all both immediate and potential issues.

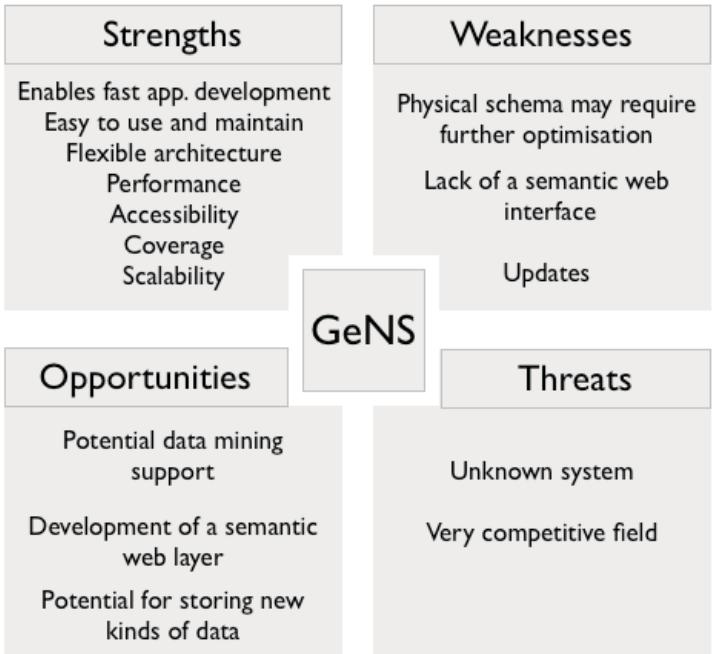

Figure 5.1 - GeNS' SWOT analysis

5.3. Future Work

GeNS is able to accommodate heterogeneous biological data under a simple yet flexible schema, thus providing a unified view of the data and boosting the process of knowledge discovery. However, as seem in Figure 5.1, several aspects of this system still need to be refined.

Firstly, the physical schema of the local database needs to be refined in order to fully and correctly integrate certain kinds of data (e.g. gene expression data from ArrayExpress Atlas). Given the characterists of heterogeneous data, this problem will likely grow larger as new, complex types of data emerge. In addition, the direct relationships from one biological entity to multiple others are an interesting set of information that should also be mapped. For both entries, the solution lies in increasing the level of data normalization – the types of data in the description table should be added to the DataType table. As for the direct relationships, a correlation table would clearly solve the issue while following good normalization practices.

Secondly, more work should be placed regarding updating the data sets. While a complex task for the UniProt set (due to its very large size), a differential update strategy could be implemented by associating a timestamp with each protein; during the update procedure, the

timestamp would be compared to the one provided by UniProt, restricting the update to the affected proteins only. This would ensure the consistency of the database even for applications running on top of GeNS that have statically referenced the internal identifiers.

Thirdly, despite the fact that the applications already using GeNS already furnish a front-end to the platform, the lack of a Web Semantics layer places GeNS behind other third-party applications (e.g. Bio2RDF). Implementing this layer would enable GeNS users' to pose expressive queries that empower data contextualization even further. This would facilitate the process of extracting new conclusions through larger and more complex sets of data confering a significant competitive advantage. To do so, a method to RDFize the integrated data should be explored in a nearby future.

Moreover, new sources of data should be inserted in order to extend the already existing coverage. Integrating more variation related data is a particularly interesting option (due to its current low volume in the database); NCBI's dbSNP [73] and PharmGKB are two prime candidates for this task, even though the latter is already being partially integrated via UniProt's data set. Being able to provide genotype and phenotype interaction data is another interesting prospect that could easily be achieved by integrating NCBI's dgGaP [74], as is the possibility of adding even further sets of data from KEGG; KEGG Disease's data, for example, would be a perfect complement to the existing data.

Finally, with such a large amount of data present in the database, being able to perform data mining operations over it may provide an interesting feature to GeNS' users due to the potential connections that may be revealed by them. For example, implementing a post-integration data coverage amplifier based on the theoretically demonstrated method described in section 3.5.5 would bestow GeNS with a unique feature that would automatically translate into a noteworthy competitive advantage. Another option would be to run a BLAST [75] algorithm over the integrated data, for example.

References

1. Sanger, F., S. Nicklen, and A.R. Coulson, *DNA sequencing with chain-terminating inhibitors.* Proceedings of the National Academy of Sciences, 1977. **74**(12): p. 5463-5467.

2. Mullis, K., et al., *Specific enzymatic amplification of DNA in vitro: the polymerase chain reaction. 1986.* Biotechnology (Reading, Mass.), 1992. **24**: p. 17.

3. Hogeweg, P., *Simulating the growth of cellular forms.* Simulation, 1978. **31**: p. 90-98.

4. Watson, J.D., *The human genome project: past, present, and future.* Science, 1990. **248**(4951): p. 44-49.

5. Wolfe, K.H. and W.H. Li, *Molecular evolution meets the genomics revolution.* nature genetics, 2003. **33**(3 s): p. 255-265.

6. Galperin, M.Y., *The Molecular Biology Database Collection: 2008 update.* Nucleic Acids Res, 2007.

7. Benson, D.A., et al., *GenBank.* Nucleic Acids Res, 2007. **35**(Database issue): p. D21-5.

8. Kanehisa, M., et al., *KEGG for linking genomes to life and the environment.* Nucleic Acids Res, 2007.

9. Parkinson, H., et al., *ArrayExpress--a public database of microarray experiments and gene expression profiles.* Nucleic Acids Res, 2007. **35**(Database issue): p. D747-50.

10. Lacroix, Z., *Biological data integration: wrapping data and tools.* IEEE Trans Inf Technol Biomed, 2002. **6**(2): p. 123-8.

11. Louie, B., et al., *Data integration and genomic medicine.* J Biomed Inform, 2007. **40**(1): p. 5-16.

12. Stein, L.D., *Integrating biological databases.* Nat Rev Genet, 2003. **4**(5): p. 337-45.

13. Topaloglou, T., A. Kosky, and V. Markowitz, *Seamless integration of biological applications within a database framework.* Proc Int Conf Intell Syst Mol Biol, 1999: p. 272-81.

14. Wong, L., *Technologies for integrating biological data.* Brief Bioinform, 2002. **3**(4): p. 389-404.

15. Zhong, W. and P.W. Sternberg, *Automated data integration for developmental biological research.* Development, 2007. **134**(18): p. 3227-38.

16. Brusic, V., et al. *Data learning: understanding biological data*. 1998.

17. KEGG. [cited 05-06-2009]; Available from: http://kegg.jp/kegg/pathway/map/map00020.html.

18. Achard, F., G. Vaysseix, and E. Barillot, *XML, bioinformatics and data integration.* Bioinformatics, 2001. **17**(2): p. 115-25.

19. Bairoch, A., et al., *The universal protein resource (UniProt).* Nucleic acids research, 2005. **33**(Database Issue): p. D154.

20. Wu, C.H., et al., *The Universal Protein Resource (UniProt): an expanding universe of protein information.* Nucleic Acids Res, 2006. **34**(Database issue): p. D187-91.

21. Kanehisa, M. and S. Goto, *KEGG: Kyoto encyclopedia of genes and genomes.* Nucleic acids research, 2000. **28**(1): p. 27.

22. Ashburner, M., et al., *Gene ontology: tool for the unification of biology. The Gene Ontology Consortium.* Nat Genet, 2000. **25**(1): p. 25-9.

23. Linnaeus, C., *Systema naturae. vol. 1.* Stockholm, 824 pp, 1758.

24. Luscombe, N.M., D. Greenbaum, and M. Gerstein, *What is bioinformatics? A proposed definition and overview of the field.* Methods of information in medicine, 2001. **40**(4): p. 346-358.

25. Letovsky, S., *Beyond the information maze.* Journal of Computational Biology, 1995. **2**(4): p. 539-546.

26. Durinck, S., et al., *BioMart and Bioconductor: a powerful link between biological databases and microarray data analysis*. 2005, Oxford Univ Press. p. 3439-3440.

27. Brazma, A., et al., *ArrayExpress--a public repository for microarray gene expression data at the EBI.* Nucleic acids research, 2003. **31**(1): p. 68.

28. Sabharwal, H.S., T.C. Inc, and C.A. Cupertino, *SQL Access and ISO/RDA.* Compcon Spring'91. Digest of Papers: p. 123-126.

29. Lee, T.J., et al., *BioWarehouse: a bioinformatics database warehouse toolkit.* BMC Bioinformatics, 2006. **7**: p. 170.

30. Chun, T.Y., *World Wide Web robots: an overview.* Online & CD-ROM Review, 1999. **23**(3): p. 135-42.

31. Reichhardt, T., *It's sink or swim as a tidal wave of data approaches.* Nature, 1999. **399**(6736): p. 517.

32. Haas, L.M., et al., *DiscoveryLink: A system for integrated access to life sciences data sources.* IBM Systems Journal, 2001. **40**(2): p. 489-511.

33. Oliveira, J.L., et al. *DiseaseCard: A Web-Based Tool for the Collaborative Integration of Genetic and Medical Information*. 2004: Springer.

34. Wiederhold, G., *Mediators in the architecture of future information systems.* Computer, 1992. **25**(3): p. 38-49.

35. Cadag, E., et al., *Biomediator data integration and inference for functional annotation of anonymous sequences.* Pac Symp Biocomput, 2007: p. 343-54.

36. Kohler, J., S. Philippi, and M. Lange, *SEMEDA: ontology based semantic integration of biological databases.* Bioinformatics, 2003. **19**(18): p. 2420-7.

37. Inmon, W.H., *Building the data warehouse*. 2005: Wiley.

38. Birkland, A. and G. Yona, *BIOZON: a hub of heterogeneous biological data.* Nucleic Acids Res, 2006. **34**(Database issue): p. D235-42.

39. Cherkasova, L., V. Kotov, and T. Rokicki. *On scalable net modeling of OLTP*. 1993.

40. Chaudhuri, S. and U. Dayal, *An overview of data warehousing and OLAP technology.* ACM Sigmod Record, 1997. **26**(1): p. 65-74.

41. Sen, A. and A.P. Sinha, *A comparison of data warehousing methodologies.* 2005.

42. Codd, E.F., *A relational model of data for large shared data banks.* 1970.

43. Allan, R.G., *The Impact of the OLAP/OLTP Cultural Conflict on Data Warehousing.* BUSINESS INTELLIGENCE JOURNAL, 2004. **9**: p. 21-26.

44. Küntzer, J., et al., *BN++-A Biological Information System.* J Integr Bioinformatics, 2006. **3**(2): p. 34.

45. Bhandarkar, M., et al., *BioCoRE: a collaboratory for structural biology.* Urbana. **51**: p. 61801.

46. Pruitt, K.D., et al., *Introducing RefSeq and LocusLink: curated human genome resources at the NCBI.* Trends in Genetics, 2000. **16**(1): p. 44-46.

47. Karp, P.D., et al., *Expansion of the BioCyc collection of pathway/genome databases to 160 genomes.* Nucleic acids research, 2005. **33**(19): p. 6083.

48. Shaker, R., et al. *The biomediator system as a tool for integrating biologic databases on the web*. 2004.

49. Kasprzyk, A., et al., *EnsMart: a generic system for fast and flexible access to biological data.* Genome Res, 2004. **14**(1): p. 160-9.

50. Oinn, T., et al., *Taverna: a tool for the composition and enactment of bioinformatics workflows*. 2004, Oxford Univ Press. p. 3045-3054.

51. Shannon, P., et al., *Cytoscape: a software environment for integrated models of biomolecular interaction networks*. 2003, Cold Spring Harbor Laboratory Press. p. 2498-2504.

52. Gentleman, R., et al., *Bioconductor: open software development for computational biology and bioinformatics.* Genome biology, 2004. **5**(10): p. R80.

53. Stevens, R., et al., *TAMBIS: transparent access to multiple bioinformatics information sources.* Bioinformatics, 2000. **16**(2): p. 184-5.

54. Dijkstra, E.W., *On the role of scientific thought.* Selected Writings on Computing: A Personal Perspective, 1982: p. 60-66.

55. Detours, V., et al., *Integration and cross-validation of high-throughput gene expression data: comparing heterogeneous data sets.* FEBS Lett, 2003. **546**(1): p. 98-102.

56. Hamosh, A., et al., *Online Mendelian Inheritance in Man (OMIM), a knowledgebase of human genes and genetic disorders.* Nucleic acids research, 2005. **33**(Database Issue): p. D514.

57. Sayers, E.W., et al., *Database resources of the National Center for Biotechnology Information.* Nucleic acids research, 2008.

58. NCBI. May 15, 2009 [cited May 17, 2009]; Available from: http://www.ncbi.nlm.nih.gov/pubmed/.

59. Bairoch, A. and R. Apweiler, *The SWISS-PROT protein sequence data bank and its supplement TrEMBL.* Nucleic acids research, 1997. **25**(1): p. 31.

60. George, D.G., W.C. Barker, and L.T. Hunt, *The protein identification resource (PIR).* Nucleic acids research, 1986. **14**(1): p. 11.

61. Pedro Lopes, D.P., D. Campos, and J. L. Oliveira, *Arabella: A Directed Web Crawler (accepted)*, in *International Conference on Knowledge Discovery and Information Retrieval (KDIR 2009)*. 2009: Madeira, Portugal.

62. Meier, J.D., et al., *Improving. NET application performance and scalability*. 2004, Microsoft Press.

63. Bergman, J., *Does gene duplication provide the engine for evolution?* Journal of Creation, 2006. **20**(1): p. 99-104.

64. Zhang, J., *Evolution by gene duplication: an update.* Trends in Ecology & Evolution, 2003. **18**(6): p. 292-298.

65. Microsoft. *Partitioned Tables and Indexes in SQL Server 2005*. 2005 [cited 29-05-2009]; Available from: http://msdn.microsoft.com/en-us/library/ms345146.aspx#sql2k5parti_topic14.

66. McDonald, S., *SQL Injection Walkthrough. White paper, SecuriTeam, May 2002*.

67. Anley, C., *Advanced SQL injection in SQL server applications.* White paper, Next Generation Security Software Ltd, 2002.

68. Arrais, J., J.G.L.M. Rodrigues, and J.L. Oliveira, *Improving Literature Searches in Gene Expression Studies*, in *Advances in Intelligent and Soft Computing : 2nd International Workshop on Practical Applications of Computational Biology and Bioinformatics*, J.M. Corchado, et al., Editors. 2009, Springer Berlin / Heidelberg: Berlin, DE. p. Capt. 10, p. 74 - 82.

69. Arrais, J., et al. *GeneBrowser: an approach for integration and functional classification of genomic data*. 2007.

70. Belleau, F., et al., *Bio2RDF: Towards a mashup to build bioinformatics knowledge systems.* Journal of biomedical informatics, 2008. **41**(5): p. 706-716.

71. Klyne, G., J.J. Carroll, and B. McBride, *Resource description framework (RDF): Concepts and abstract syntax.* W3C recommendation, 2004. **10**.

72. Prud'hommeaux, E. and A. Seaborne, *SPARQL Query Language for RDF. W3C Candidate Recommendation 6 April 2006*.

73. Sherry, S.T., et al., *dbSNP: the NCBI database of genetic variation.* Nucleic acids research, 2001. **29**(1): p. 308.

74. Mailman, M.D., et al., *The NCBI dbGaP database of genotypes and phenotypes.* Nature genetics, 2007. **39**(10): p. 1181-1186.

75. Altschul, S.F., et al., *Basic local alignment search tool.* J. mol. Biol, 1990. **215**(3): p. 403-410.