POLITECNICO DI MILANO

Scuola di Ingegneria Industriale
e dell'Informazione
Laurea in Ingegneria Informatica
Dipartimento di Elettronica, Informazione e
Bioingegneria

Design and implementation of automatic procedures to import and integrate data in a genomic and proteomic data warehouse

Advisor: Prof. Marco Masseroli, Ph.D

Co-advisor: Arif Canakoglu, MS

Master Graduation Thesis:
Vincenzo Di Girolamo
ID Number: 783097

Academic Year 2013-2014

POLITECNICO DI MILANO

Scuola di Ingegneria Industriale
e dell'Informazione
Laurea in Ingegneria Informatica
Dipartimento di Elettronica, Informazione e
Bioingegneria

Design and implementation of automatic procedures to import and integrate data in a genomic and proteomic data warehouse

Thesis by: ____________________
(Vincenzo Di Girolamo)

Advisor: ____________________
(Prof. Marco Masseroli)

Academic Year 2013-2014

A Giada, Lina

e Mario

Acknowledgements

First, I would like to thank the advisor, Prof. Marco Masseroli, for his valuable teachings and for having always supported my work with passion, stimulating my interest in Bioinformatics.
I also want to thank Arif Canakoglu for the valid and constant help that he has given me in the technical aspects of this Thesis and for being a friend as well as a guide.

Table of contents

List of figures III

List of tables V

Abstract / Sommario VI

1 Introduction 1

2 State of the art 4

2.1 Genomic and proteomic research 4

2.2 Controlled vocabularies, ontologies and functional annotations 5

2.3 Biomolecular databanks 7

2.4 Data file formats 11

2.5 Difficulties in effectively using of available biomolecular data 12

3 Thesis goals 14

4 Genomic and Proteomic Data Warehouse (GPDW) software framework 16

4.1 Used software and tools 16

4.2 GPDW integrated data model 16

4.3 GPDW framework description 18

4.4 Data import procedures 19

4.5 Data integration procedures 22

4.6 Metadata computation and storage 24

5 GPDW framework enhancements 26

5.1 Enhancement of GPDW_definition.xml configuration file 27

5.2 Enhancement of feature_definitons.xml configuration file 30

5.3 Automatic import of source tables 33

5.3.1 Generic main source tables Loader 36

5.3.2 Generic additional source tables Loader 37

5.3.3 Relationship tables Loader 38

5.4 Post-processing and data recovery components 40

5.4.1 Post-processing 44

5.4.2 Data recovery 45

6 Considered data sources 46

6.1 Gene Ontology - GO 47

6.2 Gene Ontology Annotation - GOA 51

6.3 Entrez Gene 55

6.4 Expert Protein Analysis System ENZYME - ExPASy ENZYME 58
6.5 Online Mendelian Inheritance in Man - OMIM 61
7 Implementation of data import automatic procedures 67
7.1 Parser implementation for GO go_daily-termdb.obo-xml data file 67
7.2 Import of source tables 73
7.2.1 ExPASy ENZYME enzyme.dat data file 73
7.2.2 GOA gene_association.goa_<species> data files 76
7.3 Post-processing and data recovery operations on import omim.txt data file 79
8 Validation and testing 86
8.1 Imported data errors and inconsistencies 86
8.1.1 GO 86
8.1.2 GOA 87
8.1.3 Entrez Gene 88
8.1.4 ExPASy ENZYME 89
8.1.5 OMIM 89
8.2 Quantification of imported data and running time 90
9 Final discussions and conclusions 94
10 Future developments 96
References 97

List of figures

Figure 1 – Moore's law 7
Figure 2 – Internet domain survey host count 7
Figure 3 – Databanks growth 9
Figure 4 – Biomolecular databanks and their relations 10
Figure 5 – Genomic data file formats 11
Figure 6 – Tabular file types 11
Figure 7 – XML file types 12
Figure 8 – General conceptual model for feature from different sources 17
Figure 9 – GPDW software architecture components....18
Figure 10 – Sequence diagram of import process 20
Figure 11 – Sequence diagram of integration process 23
Figure 12 – metadata.source2feature_association database table. 25
Figure 13 – GPDW framework import layer components....26
Figure 14 – Example of source file definition 28
Figure 15 – XML Schema description of feature definition....29
Figure 16 – Example of feature association definition....30
Figure 17 – Similarity integrated table template 31
Figure 18 – Source tables definition of ExPASy ENZYME databank 32
Figure 19 – Templates for imported additional source tables....32
Figure 20 – GenericLoader workflow - part I 34
Figure 21 – GenericLoader workflow - part II....35
Figure 22 – Method insertSourceTableEntry 38
Figure 23 – Relationship imported table template 39
Figure 24 – Example of query generated by RelationshipDataLoader....39
Figure 25 – ImportManager workflow - part I 41
Figure 26 – ImportManager workflow - part II 42
Figure 27 – ImportManager workflow - part III....43
Figure 28 – Example of GO gene product annotation....48
Figure 29 – GO ER schema 49
Figure 30 – GO Logical schema - cellular component source tables 50
Figure 31 – GO Logical schema - association tables....50
Figure 32 – GO Logical schema - integrated source tables....51

Figure 33 – Example of records in file pfam2go....52
Figure 34 – GOA ER schema....54
Figure 35 – GOA Logical schema....54
Figure 36 – Query results for human muscular dystrophy in Entrez Gene....55
Figure 37 – Example of records in file gene2go....56
Figure 38 – Entrez Gene ER schema....57
Figure 39 – Entrez Gene Logical schema....57
Figure 40 – Example of records in file enzclass.txt....58
Figure 41 – ExPASy ENZYME ER schema....59
Figure 42 – ExPASy ENZYME Logical schema - imported tables....60
Figure 43 – ExPASy ENZYME Logical schema - integrated tables....60
Figure 44 – OMIM ER schema - part I....62
Figure 45 – OMIM ER schema - part II....63
Figure 46 – OMIM Logical schema - gene....64
Figure 47 – OMIM Logical schema - genetic disorder....65
Figure 48 – OMIM Logical schema - clinical synopsis....66
Figure 49 – OMIM Logical schema - association tables....66
Figure 50 – go_daily-termdb.obo-xml header....68
Figure 51 – OBO ontology term....69
Figure 52 – Example of GO obsolete term....70
Figure 53 – Example of GO synonym term....71
Figure 54 – Example of GO term's relationship....72
Figure 55 – Example of records in file enzyme.dat....73
Figure 56 – Entry creation for expasy_enzyme table....75
Figure 57 – Entry creation for expasy_enzyme_action table....75
Figure 58 – Example of records in file gene_association.goa_human....78
Figure 59 – Entry creation for GOA association tables....78
Figure 60 – Base xml structure for OMIM clinical synopsis normalization....79
Figure 61 – log.omim_history_tmp table....84
Figure 62 – System log warning messages for omim.txt....84
Figure 63 – SQL query for OMIM relationship tables....85
Figure 64 – SQL query for gene2genetic_disorder_imported....85
Figure 65 – biological_function_feature2pathway_imported table....86
Figure 66 – SQL queries for GOA secondary association table....87
Figure 67 – Proposed solution for GOA secondary association tables....88

List of tables

Table 1 – List of features considered in GPDW 17
Table 2 – Table of colors and types of Logical schemas tables 47
Table 3 – Associations in gene_association.goa_uniprot 53
Table 4 – Number of entries in OMIM 61
Table 5 – Field labels in file enzyme.dat 73
Table 6 – Field labels in file omim.txt 80
Table 7 – Total number of imported entries and running time 90
Table 8 – Details of imported tables of GO 91
Table 9 – Details of imported tables of GOA 91
Table 10 – Details of imported tables of Entrez Gene 92
Table 11 – Details of imported tables of ExPASy ENZYME 92
Table 12 – Details of imported tables of OMIM 93

Abstract

The growing of information technologies and biotechnologies provides new scenarios for novel research approaches and greatly influences the evolution of modern disciplines as Bioinformatics.
New biomedical applications, providing effective data management and analysis support, allow the integration and evaluation of controlled data with the goal of unveil new biomedical knowledge. Data warehousing is the main significant approach used in data integration when, as in the case of genomics and proteomics, transformation is required to clean data and make them available for querying and integrated analysis.
Bioinformatics highlights the relevance of using computational technologies to describe and analyze biological systems in order to formulate hypothesis about life's molecular processes. The goal of Bioinformatics is to organize databases, analyze the acquired knowledge about genome and proteome and finally store, retrieve, visualize and effectively evaluate the available data and information.
Today there are several public biomolecular databanks that offer to biologists, physicians and researchers the possibility of online consultation and download of such data freely. However these data are very heterogeneous and distributed, so that it is quite hard to create a consistent global overview of them. Therefore it is needed to have computational tools that overcome cross-search problem and provide the information not directly available from individual data sources. Already proposed models are quite complex, mainly suitable for single organisms and, in many cases, they require notable maintenance effort, in particular when data evolve rapidly, also in their structure.
For this purpose, the Bioinformatics and Web Engineering Lab of Politecnico di Milano is working on a project, named Genomic and Proteomic Data Warehouse (GPDW), in order to create a data warehouse that integrates distributed information from many sources of genomic and proteomic data so that the integrated information is frequently updated, ensuring the quality of available biomolecular data integrated.
The primary goal of this Thesis is to extend and generalize the process of the creation of the data warehouse, by creating new generic and automatic procedures for data extraction, transformation and load. The second Thesis goal is to implement necessary operations to integrate data, from the considered databanks, to the GPDW project, by developing generic components to be integrated to the existing GPDW software architecture.
GPDW framework is based on a flexible multi-level data model which includes a source-import lower tier, an instance-aggregation middle tier and a concept-integration upper tier. This model is composed of interconnected modules, representing biomolecular entities and their biomedical features.
A conceptual module, or feature, is structured in two levels, import and aggregation levels, whose concrete data computation is realized by data import and integration automatic

procedures. The supporting structure of metadata provides a complete description of the conceptual data model, and eases data traceability, validation and consistency.
Relevant aspects of Computer Science and Software Engineering are actually considered in the design of a new set of abstract procedures. The integration of new components to the framework extends generalization and modularity properties; in the same time, automatic procedures are able of adapting to small changes in data formats and to the continuous evolution of data in integrated sources.
The customization of data parsing and loading procedures is applied to the considered data sources through the extension of the generic procedures defined and the specification of suitable metadata; additional operations of post-processing and data recovery are implemented, when closely required, by extending and customize generic procedures defined for this purpose, in order to complete the import process effectively.
Testing and consistency checking are designed to validate the data imported in the data warehouse, by highlighting errors and anomalies, which trigger evaluations on possible enhancements and future developments.

Sommario

Lo sviluppo delle tecnologie informatiche e delle biotecnologie offre nuovi scenari per un nuovo approccio nella ricerca scientifica e influenza l'evoluzione delle moderne discipline come la Bioinformatica.
Le nuove applicazioni biomediche, che forniscono un efficace supporto all'analisi e alla gestione delle informazioni, consentono l'integrazione e la valutazione di dati controllati con lo scopo di svelare nuova conoscenza biomedica. A tal proposito, il data warehousing è stato significativamente usato per integrare dati che richiedono, come nel caso della genomica e della proteomica, di essere ripuliti prima di poter essere interrogati e sottoposti ad una analisi complessiva.
La Bioinformatica evidenzia l'importanza di utilizzare le tecnologie computazionali per descrivere e analizzare i sistemi biologici allo scopo di formulare ipotesi sui processi molecolari della vita. Il fine della Bioinformatica è organizzare banche dati, analizzare le conoscenze acquisite sul genoma e sul proteoma e, per finire, conservare, recuperare, visualizzare e valutare efficientemente i dati e le informazioni disponibili.
Oggi, esistono diverse banche dati biomolecolari, liberamente accessibili sul Web, che offrono a biologi, medici e ricercatori la possibilità di consultare online e scaricare tali dati. Tuttavia questi dati sono molto eterogenei e dispersi così che è abbastanza difficile creare una coerente visione d'insieme di essi. Nasce quindi la necessità di avere strumenti informatici per eseguire ricerche incrociate e recuperare le informazioni che non sono direttamente disponibili nelle diverse sorgenti considerate singolarmente. I modelli che sono stati proposti risultano complessi, principalmente adatti a descrivere singoli organismi e, nella maggior parte dei casi, richiedono un notevole lavoro di manutenzione, soprattutto quando i dati evolvono, anche nella loro struttura.
In tal senso, il laboratorio di Bioinformatics and Web Engineering del Politecnico di Milano sta lavorando alla realizzazione di un progetto denominato Genomic and Proteomic Data Warehouse (GPDW), con l'obiettivo di creare un data warehouse che integri le informazioni distribuite su molte sorgenti di dati genomici e proteomici, in modo che i dati integrati siano frequentemente aggiornati e validati.
Il primo scopo di questa Tesi è quello di estendere e generalizzare il processo di creazione del data warehouse, realizzando nuove procedure automatiche per l'estrazione, la trasformazione e caricamento dei dati. Il secondo obiettivo è quello di implementare le operazioni necessarie per l'integrazione dei dati, forniti dalle banche dati considerate, nel progetto GPDW, sviluppando nuovi moduli software, da integrare ai componenti presenti nell'architettura framework del GPDW.
Il framework GPDW si basa su un modello dei dati flessibile e multilivello che include uno strato inferiore di importazione, uno intermedio di aggregazione e uno superiore di

integrazione concettuale. Questo modello si compone di moduli interconnessi, rappresentati dalle entità biomolecolari e dalle loro caratteristiche biomedicali.
Un modulo concettuale, o feature, è strutturato su due livelli, di importazione e di integrazione, la cui concreta realizzazione è operata dalle procedure automatiche di importazione e integrazione dei dati. La definizione dei metadati, inoltre, fornisce una completa descrizione dello schema concettuale dei dati, facilitando la loro tracciabilità, validazione e consistenza.
Gli aspetti rilevanti dell'Informatica e dell'Ingegneria del Software sono concretamente considerati nel design di un nuovo insieme di procedure astratte. L'integrazione di nuovi componenti al framework estende i concetti di generalizzazione e modularità; inoltre, le procedure automatiche restano valide in caso di piccoli cambiamenti nel formato dei dati o del continuo aggiornamento dei dati nelle sorgenti integrate.
La personalizzazione delle procedure di parsing e caricamento dei dati è realizzata per le sorgenti dati considerate, mediante l'estensione delle procedure generiche definite e la specificazione di adeguati metadati; aggiuntive procedure di post-processing e recupero dati sono implementate, quando strettamente necessario, estendendo e personalizzando le generiche procedure definite a questo scopo, in modo da completare il processo di importazione in modo corretto.
Verifiche di consistenza e test sono progettati per validare i dati importati nel data warehouse, evidenziando anomalie ed errori che diventano spunti di riflessione per possibili miglioramenti e sviluppi futuri.

1. Introduction

Bioinformatics is the application of statistics and computer science to the field of molecular biology [1].
Bioinformatics involves the use of techniques and concepts from mathematics, informatics, statistics, artificial intelligence, chemistry, biochemistry and physics to solve problems in biology, most commonly molecular biology. In general, the aims of Bioinformatics are:

- Provide qualified statistical models for understanding the meaning of data from biochemistry and molecular biology experiments in order to unveil common patterns and identify general laws;
- Propose new models and mathematical tools to analyze DNA, RNA and protein sequences with the goal of creating a set of knowledge about frequent sequences, their evolution and their functions;
- Organize data and global knowledge about genome and proteome into curated and computationally inferred databases in a way that allows researchers to easily access to existing information, submit new entries and retrieve meaningful data.

Traditionally, Bioinformatics concerned to individual systems, studying RNA and DNA sequences in detail. The evolution of Bioinformatics has led to a pervasive use of computer sciences in biology and the birth of a new discipline called Computational Biology, which properly explains the strong connection between biology and information technologies.
Over the past few decades, major progresses in the field of molecular biology, coupled with advances in genomic technologies, have caused an explosive growth in the biological information generated by the scientific community.
This amount of genomic information has, in turn, led to an absolute requirement for computerized databases to store, organize, and index the data and for specialized tools to retrieve and analyze these data [2].
A biological database is a large, organized container of persistent data, usually associated with computerized software designed to update, query, and retrieve information stored within the system. A simple database might be a single file containing many records, each of which includes the same set of information.
For example, a record associated with a nucleotide sequence database typically contains information such as contact's name, input sequence, description of the molecule's type, scientific name of the source organism from which it was isolated and, often, literature citations associated with the sequence.
Thus Bioinformatics is the scientific field in which biology, computer science, and information technology merge into a single discipline; its ultimate goal is to enable the discovery of new biological insights as well as to create a global perspective from which

unifying biological principles can be discerned. At the beginning of the genomic revolution, Bioinformatics aimed to create and maintain databases to store biological information, such as nucleotide and amino acid sequences.
This type of databank requires facing design and implementation issues but also the development of complex interfaces whereby biologists and researchers could both access existing data as well as submit new or revised ones.
In order to study how normal cellular activities are altered in different disease states, the biological data must be combined to give a comprehensive picture of these activities. Hence, Bioinformatics has evolved in the direction of extending the analysis to various different types of data, including nucleotide and amino acid sequences, protein domains, and protein structures.
The actual process of analyzing and interpreting data is referred to as Computational Biology. Important sub-disciplines within Computational Biology include:

- Development and implementation of tools that enable efficient access, usage and management of various types of information;
- Development of new algorithms and statistics to discover relationships among members of large data sets, e.g. methods to locate a gene within a sequence, to predict protein structure and/or function, and to cluster protein sequences into families of related sequences.

Today, the first challenge facing the bioinformatics community is the intelligent and efficient storage of this mass of information, besides of the responsibility to provide easy and reliable access to these data.
The data itself is meaningless before the analysis and its huge volume makes it impossible, even for a trained biologist, any manual interpretation. Therefore, powerful computer tools must be developed to allow the extraction of meaningful biological information. There are three central biological processes (central dogma of Bioinformatics) around which bioinformatics tools must be developed:

- DNA sequence determines protein sequence;
- Protein sequence determine protein structure;
- Protein structure determines protein function.

The integration of information learnt about these key biological processes should allow the achievement of the long term goal represented by the complete understanding of organisms' biology.

Following the introduction, Chapter 2 is about the conceptual meaning of Bioinformatics, with the origins and the historical development of the discipline and its main tasks and goals. Information about genomic and proteomic fields is briefly explained, together with the importance of information technologies in biomedical and biomolecular research.

Then the chapter deals with controlled vocabularies, ontologies, functional annotations and their usage in Bioinformatics. In the end, the chapter presents an overview on biomolecular databanks that are available online and data types and formats that they provide.

The goals of the thesis are discussed in Chapter 3.

Chapter 4 starts with the analysis of the data model used in GPDW framework. The main components that manage the entire process of creation of the data warehouse are described. Afterwards, the activities required for the whole process of data importing, integration and metadata computation and storing into the data warehouse are discussed, pointing out their design limits.

Chapter 5 contains the description of the automatic procedures for data importing and integration. The design choices, related to the relevant aspects of computer science, and the workflows of GPDW framework processes are illustrated. In details, the first part is about the enhancements in the configuration files; the second part describes the design and implementation of new components devoted to the source tables data importing; the third part focuses on the implementation of post-processing and data recovery generic and customizable procedures, used to manage specific sources issues.

In Chapter 6, the databanks considered in this Thesis are presented by explaining some general, historical and statistical information. For each data source it is also provided the designed ER Schema and Logical Schema containing the description of imported and integrated tables.

Chapter 7 describes use cases of the enhancements to the software architecture; the content of the files that are objects of the analysis is briefly explained and the design choices and strategies adopted to achieve a correct and consistent import are illustrated.

Chapter 8 underlines errors and inconsistencies of imported data; the chapter also shows some quantitative results related to the imported data and the time taken to import them.

In Chapter 9, the final discussion shows the conclusions and the summary of achieved results, which confirm the legitimacy of the design choices and activities undertaken to achieve the goals.

Chapter 10 proposes suggestions for future developments.

In conclusion, it is shown the list of references to books, scientific papers and web sites referred in the elaboration of this Thesis.

2. State of the art

The goal of this chapter is to describe the conceptual meaning of Bioinformatics and the importance of information and communication technologies in biomedical and biomolecular investigation, with particular attention to the main aspects of genomic and proteomic research and to the instruments used in data analysis and management
Finally the chapter presents an overview on biomolecular databanks that are available online and data types and formats that they provide.

2.1 Genomic and proteomic research

Genomics is a discipline in genetics concerning the study of the genome of organisms. In particular, it focuses on the structure, content, functions and evolution of genome.
It is based on Bioinformatics in order to process and display enormous amount of data and it is strictly connected to Molecular Biology techniques, such as gene cloning and DNA sequencing.
The complete knowledge of organisms' genome facilitates a new approach in biological research based on in silico experiments, i.e. the replication of experimental results via computer simulation.
Thanks to the human DNA sequencing by Human Genome Project (HGP) in 2000, today it is possible to predict the incidence of certain pathology on an individual or target sample in respect to the population it comes from. Genomics has the aim of preparing genetic maps of living beings by the completion of DNA sequencing. Next it continues with DNA sequence annotation, that is the identification of all the genes and significant sequence's portions, together with their available information.
The next step in relation to genomics has led to the identification of a new discipline called proteomics. The term "proteomics" was first coined in 1997 [3] to make an analogy with genomics, the study of the genes.
The term "proteome" is a blend of "protein" and "genome" and it was coined by Marc Wilkins in 1994 to describe the entire set of protein products expressed by a genome of an organism or a biological system. Proteome differs from cell to cell and it continuously interacts with genome and evolves to answer environmental modifications.
Proteomics research evaluates protein activities, modifications and localization; it identifies the interactions of proteins in complexes and allows to correlate the level of produced proteins to the activity state of a given cell or tissue.
Unlike genome, proteome is a dynamic system, which shows radical changes that are modulated by many factors, both in physiological and pathological states. The proteome of each cellular type of an organism is unique.

Only a small part of potentially active genes is transcribed and translated in a certain type of cells, with the number of RNA copies which does not strictly correlate to the number of proteins synthesized in these cells. The factors influencing proteome variability include mRNA editing, alternative splicing, co- and post-translational modification.

The area of genomic and proteomic research concerns activities based on molecular research of transcripts and proteins expressed in a cellular compartment; molecular research involves separation of hundreds of protein products, quantification of protein products expression and investigation of biological processes basic mechanisms.

This area is very multidisciplinary and requires the integration of biochemical, bioanalytic, bioinformatics and biomolecular knowledge.

The final purpose is the development of new methods to improve selection, accuracy, and interpretation of data connected to biological signals in order to identify molecular targets involved into the gene expression alterations.

In this complex context, information technologies and computer science provide tools and analysis techniques to manage the big amount of data generated by genomic and proteomic research; these instruments organize data in a reliable and effective way to guide scientists and researchers to the evaluation and comprehension of experimental results. These instruments are represented by controlled vocabularies, ontologies and annotations.

2.2 Controlled vocabularies, ontologies and functional annotations

A controlled vocabulary is a set of selected terms that provides a method to organize knowledge into an ordered, consistent, easy readable and preferably unique way.

This set of terms must be approved by the designer, in contrast to natural language vocabularies where there are no restrictions on the lexicon; in this way, the problems of homographs, synonyms and polysemies are solved, or nearly so, reducing ambiguity inherent in human languages where the same concept is usually given by different names. Thus, controlled vocabularies enable fast access and retrieval of information by computers.

Each term in a controlled vocabulary is uniquely identified by an alphanumeric code that represents a particular concept or feature.

A common example of controlled vocabulary is represented by UMLS (Unified Medical Language System) Metathesaurus, which was created and it is maintained as a support for integration of biomedical textual annotations scattered in distinct databanks, providing a single concept connecting together all related biomedical concepts [4]. Its goals are:

- Develop computer tools that parse and understand the biomedical language;
- Facilitate the communication between different systems;
- Design information retrieval on patient record systems.

In the context of genomic annotation, controlled vocabularies can be divided into two main categories:

- Flat controlled vocabulary or terminology: the terms in the vocabulary are not structured or linked through relationships;
- Structured controlled vocabularies or ontologies: the entities in the vocabulary are structured hierarchically, from most generic (root) to the most specific (leaf) through relationships characterized by a particular semantic meaning, which represent the so called semantic network.

In the following of the thesis, often the terms gene and protein will be substituted by the expression biomolecular entity, that groups them into a unique term, when the topics will be valid for both objects. Generally, biomolecular entity will refer to all the elements that can be described through specific terms of controlled vocabularies.
The definition of term ontology is not universally accepted. According to B. Smith, "we use the term ontology in what follows to refer to any theory or system that aims to describe, standardize or provide rigorous definitions for terminologies used in the domain".
Then, ontology is a formal representation of knowledge, described in both textual and computable form, as a set of concepts within a domain and the relationships between those concepts [5]. While in the last few decades the use of ontology was limited, nowadays the number of organizations adopting ontologies exponentially increased and many flat vocabularies have been converted in structured ones.
In biological and biomolecular fields, among flat controlled vocabularies, the most popular are OMIM (Online Mendelian Inheritance in Man) for genetic diseases' description, containing a catalogue of human genes and genetic phenotypes, and Reactome for the description of human reactions and pathways [6].
The OBO (Open Biological Ontologies) project, that is part of the resources of the NCBO (National Center for Biomedical Ontology), represents an effort to create controlled vocabularies for shared use across different biological and medical domains. It establishes principles and standard rules that must be respected for ontology design and development, to reach the goal of integrating and organizing knowledge in a common, structured and well documented manner [7][8]. Examples of OBO ontologies are ChEBI (Chemical Entities of Biological Interest), which focuses on chemical components, and Gene Ontology (GO), the most famous ontology for biomolecular entities annotation, that will be well discussed in Chapter 6.
Annotation is defined as the association between a biomolecular entity and a specific term of a controlled vocabulary; which describes its characteristics. In ontology, node properties are implicitly inherited from root to leaves so ontological annotations are unfolded from leaves to root.

Functional annotations can be assigned to genes and proteins in two ways: manually (human curated) and computationally inferred. Several methods have been proposed for detecting biomolecular annotations [9][10][11]; many of them are based on predictive models and provide probabilistic predictions of functional annotations.
Other methods, instead, promote the effective semantic clustering and the simple transitive relationship approaches [12][13].

2.3 Biomolecular databanks

Since the first appearance of computers, and more in the last twenty years, the power of computers has experienced an exponential growth, in agreement with Moore's Law which states: "The performance of the processors and the number of transistors on it are doubling almost every 2 years", as shown by the graph in Figure 1.

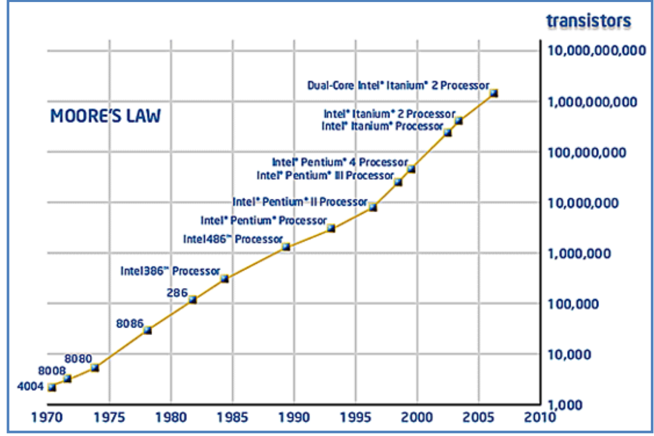


Figure 1 - Moore's law

In parallel to the rise of computational power, as a consequence of it, also Internet has suddenly evolved as proved by the graph in Figure 2.

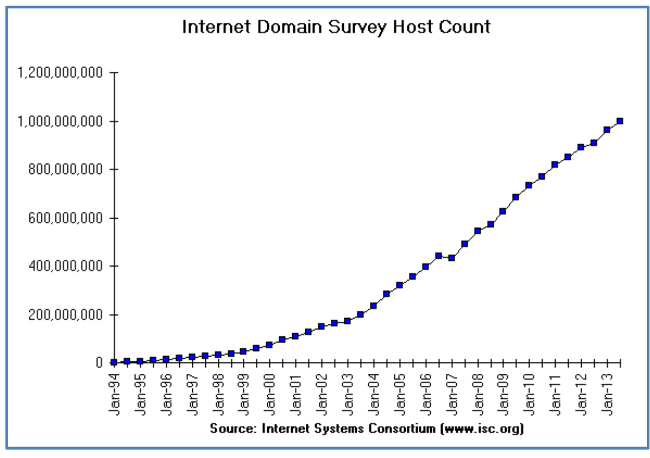


Figure 2 - Internet domain survey host count

The great growth of computers and Internet has contributed to the development of Bioinformatics. The advent of information technology caused the development of databases. The term database, or databank, means a collection of data organized to model relevant aspects in a way that supports processes requiring this information. A database is usually connected to a DBMS (Database Management System), i.e. a set of software applications that allows a unified and high-level data administration by enabling the definition, creation, querying and update of databases [14]. Biological databanks are mostly public and freely accessible [15]. They are consistent archives and contain wide spectrum data fields of molecular biology. Biological databases can be subdivided in two groups:

- Primary databanks, containing data about nucleic and amino acids sequences. These databases are defined primary because they contain only the minimal information to be associated with a certain sequence. DNA databanks include:
  - GenBank at NCBI - http://www.ncbi.nlm.nih.gov/Genbank/
  - EMBL at EBI - http://www.ebi.ac.uk/embl/
  - DDBJ - http://www.ddbj.nig.ac.jp/

  Protein databanks include:
  - UniProt - http://www.uniprot.org/
- Derivative or specialized databanks, containing information on protein families and domains, pathways, genomes, genetic disorders and literature citations. These databases collect heterogeneous data from the taxonomic and functional point of view, adding to the information available on primary databanks revised and annotated data. Examples of derivative databanks are:
  - KEGG - http://www.genome.jp/kegg/
  - PubMed - http://www.ncbi.nlm.nih.gov/PubMed/
  - GDB - http://www.gdb.org/

It is essential that these databanks are easily accessible and that they provide an intuitive query system to enable the scientific community to obtain specific information about a particular biological argument. For these reasons there were created specialized databases for particular subjects such as genomes, proteins, nucleotides, scientific literature, microarrays data and taxonomies.
As shown in Figure 3, the databases have been a large expansion during last years.

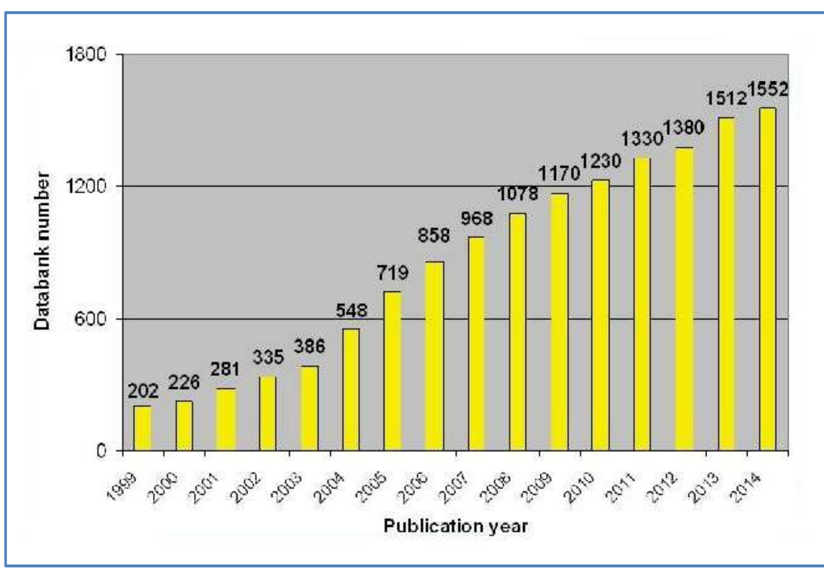


Figure 3 - Databanks growth

Databanks provide different access methods:

- Access via Web interfaces (HTML or XML): the information is usually returned as unstructured data through heterogeneous interfaces; it allows single sequence query results mainly in HTML format, thus it takes a long time to answer.
- Access via Web Services: this service is available for few databases with a limited number of items and usually it needs good computer skills to be properly exploited.
- Access via FTP server: it needs re-implementation of the database locally, requiring considerable human and computer resources.
- Direct access: it is rarely available for security issues and it highlights the lack of a common vocabulary.

Figure 4 shows the largest biomolecular databases on the Web.

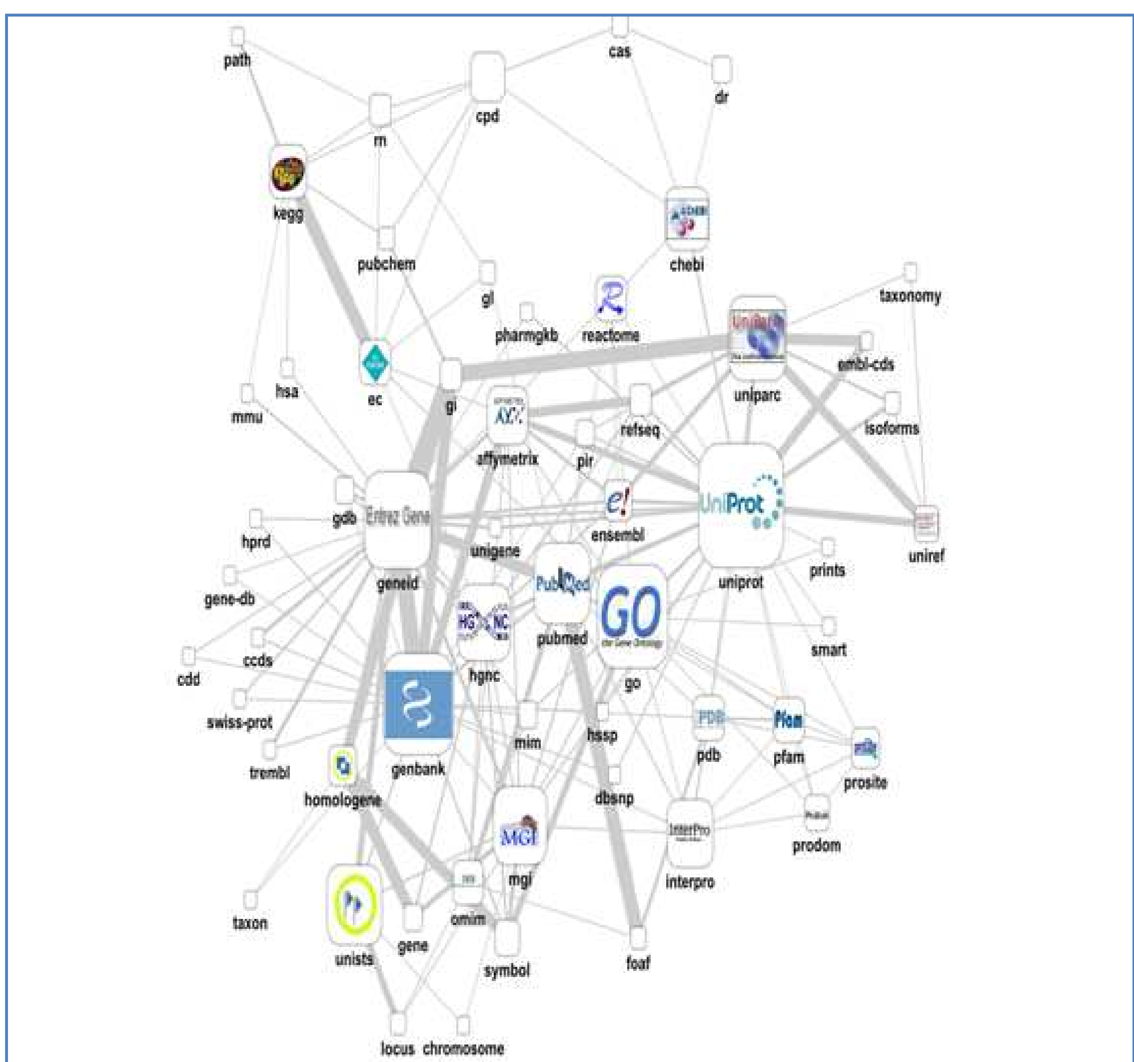


Figure 4 - Biomolecular databanks and their relations

2.4 Data file formats

The databases can provide the same data in different formats. There is no common standard for the file type and format which represent information [16]. Genomic data are usually provided with different types of files, as shown Figure 5.

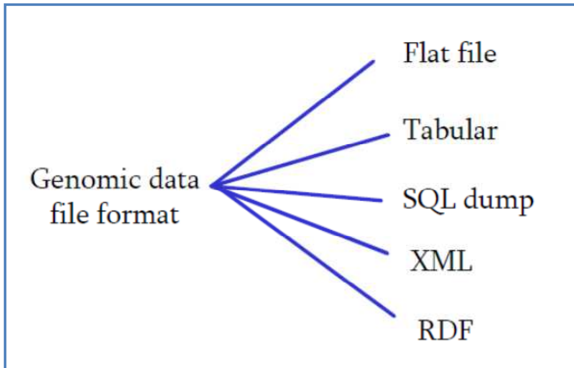


Figure 5 - Genomic data file formats

The flat file format is defined as a structured text file containing values and relations of these values. In the field of genomic data, a flat file is a text file in which each row has a different semantics, defined by the label which is inserted at the beginning of the line. It is possible that in the same line there can be two or more labels where the successive labels specify the semantics of the previous ones; if the beginning of a line is not indicated by any particular label, the line inherits the semantics of the previous line.
Tabular format contains data that are organized into rows and columns separated by one or more separators. In this case, the semantic value of a given position depends on its row and column. In this file format, if present, the header helps to understand the contents of the file or it includes statistics and other supporting information. In Figure 6 there are described the different types of existing tabular file.

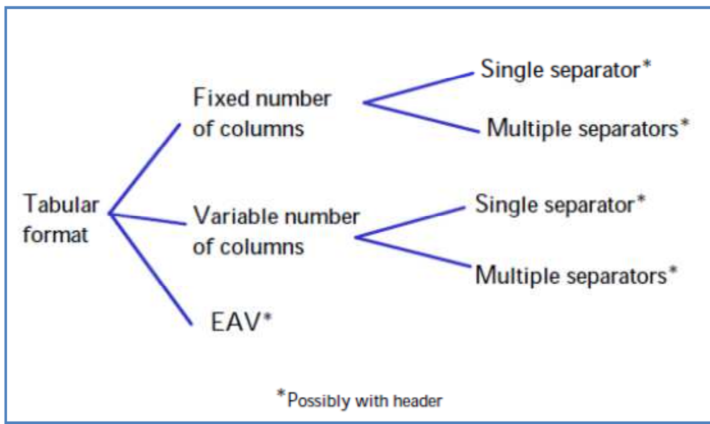


Figure 6 - Tabular file types

Some databanks, usually very specific DBMS, provide their data as SQL dump file; this format is not portable because of the lack of foreign keys and constraints in the dump. Using this format, different DBMS or earlier versions of the same DBMS cannot import data properly.
XML (eXtensible Markup Language) is a meta-language, i.e. a format to represent data, proposed by the World Wide Web Consortium (W3C) to describe and distribute structured electronic documents in the Web [17]. It is also an instrument for exporting data from heterogeneous sources.
Because of these characteristics many databases provide their data in XML format. The structure of the instance of an XML document can be described and validated by the Document Type Definition (DTD) or the XSD (XML-Schema). In Figure 7 there are described the different types of existing XML file.

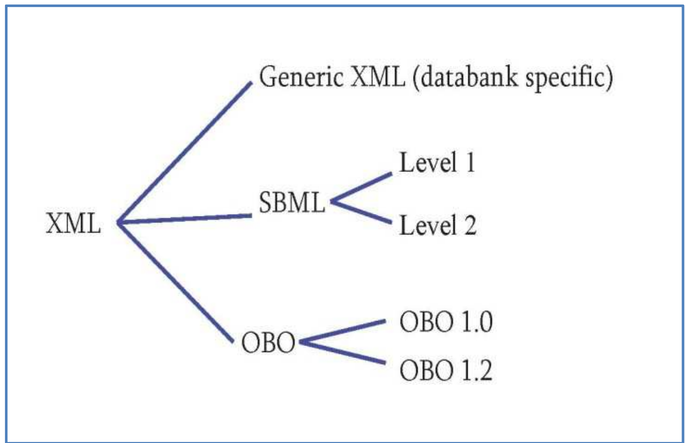


Figure 7 - XML file types

The RDF (Resource Description Framework) format is another standard XML format. It is also proposed by the W3C for the encoding, exchange and reuse of structured metadata and it provides interoperability between applications that exchange information over Web. RDF format is based on two components: RDF Model and Syntax, which describes the syntactical structure of the RDF model; RDF Schema, that shows the syntax in order to define schemas and vocabularies for metadata.

2.5 Difficulties in effectively using of available biomolecular data

According to the characteristics of biomolecular databanks previously described, it is possible to summarize the main issues concerning the usage of biological and biomolecular information.
Firstly, there are geographically distributed databases which can include redundant and overlapping data. In most cases gene and protein data are sparsely stored among

heterogeneous databanks and complex techniques for summarizing, visualizing and comparing them are required.
Consequently there exist the problems of managing the large amount of heterogeneous data and of giving to the system's users a homogeneous view as possible.
In recent years, scientific community is working on the integration of these data into a single central database. Indeed, when the data to be integrated are very numerous and off-line processing is required to efficiently and comprehensively mine the integrated data, data warehousing is the most adequate approach [18]. For this purpose, GPDW and other data integration systems have been developed. For example, BioKleisli [19] is a federated database using an object-oriented type system but it is based on a specific warehouse schema that requires to be perfectly known to be queried. K2 [20] represents the evolution of BioKleisli; it relies on GUS (Genomics Unified Schema), a complex relational database global schema that is no flexible and needs to be continuously integrated.
Then, another issue concerns the controlled vocabulary. Despite claiming to be as orthogonal to each other, there are relationships between a controlled vocabulary set and another, more in case of different organizations management. As a result of it, interoperability between different systems is still far from being achieved.
In the end, there are problems concerning the quality of annotations in databases, the difficulties faced by the curators to validate new records and the difficulty in managing and maintaining such records.
For all these reasons, ontological query answering is still problematic and specific solutions need to be found for databanks with different access methods. This thesis tries to answer, efficiently and effectively, these issues in the development of Genomic and Proteomic Data Warehouse.

3. Thesis goals

The integration of biomolecular data is an important aspect of the bioinformatics research, both for the role it has in the life sciences context research and for the challenges it requires to overcome, including efficient data storage, consistency checking and provenance checking. Biomolecular investigation allows answering questions of interest by analyzing the various types of data and information in order to obtain evidences that support results and conclusions; this investigation process contributes to the extraction of knowledge that can be used to formulate and validate hypothesis, possibly to discover new biomedical knowledge.

The exponential increment of available biomolecular information in structured or semi-structured forms, the variety, dispersion and fast evolution of such genomic and proteomic data into several distributed databanks demand continuous efforts for integration and automatic analysis methods.

The project GPDW (Genomic and Proteomic Data Warehouse) focuses on this problem. The GPDW aims not only to create a local data warehouse that easily imports and integrates data records and external references from several well-known databases, but also to keep the integrated information frequently updated, make them available to the scientific community, improve available biomolecular data quality and infer new information from available integrated data.

Indeed, GPDW allows highlighting unexpected information patterns leading to biomedical discoveries and new annotations prediction through transitive closure and semantic clustering approaches, as previously reported.

This Thesis has as primary goal to extend and completely generalize the procedure for the creation of an integrative data warehouse, in particular with reference to the extraction, transformation and load staging procedures involved in the data warehousing data integration and their clearing and quality improvement.

In the specific case of GPDW, the completion of design and implementation of data importing general procedures in the software framework by reengineering XML configuration files and by extending modules and automatic programs for data import and integration.

The second goal is to use such enhanced procedures to import and integrate data from sources that are not yet considered in the GPDW project and, at the same time, improve the procedures already integrated in the GPDW software architecture. In details, the development of the second objective requires:

- Conceptual and logical analysis and modeling of data provided by considered sources;

- Development and configuration of automatic components for the import and integration of data into the GPDW data warehouse; the developed modules are integrated with the existing components and take part in the data transformation and loading process at different stages.
- Implementation of parsing procedures, according to the specific data file format, and importing procedures to insert into the data warehouse the information extracted from considered source;
- Design of testing procedure for consistency checking and reporting of data anomalies. Analysis of metadata and reporting of the errors discovered in import data operations are used to highlight data inconsistencies.

4. Genomic and Proteomic Data Warehouse (GPDW) software framework

In the following pages, GPDW software framework architecture is presented. First the description of the flexible global data schema of GPDW framework is introduced. Then, the discussion continues with an overview on the main software packages available in the framework that manage the data warehouse creation process. Data import and integration procedures are discussed, pointing out their design limitations. The chapter terminates with a section dedicated to explain the metadata computation.

4.1 Used software and tools

GPDW framework has been developed in Java programming language using Eclipse, an open-source integrated development environment (IDE) that contains a base workspace and an extensible plug-in system for customizing the environment in order to manage all the software life cycle.
PostgreSQL is the software used for the data structure; it is a popular and reliable object-relational database management system (ORDBMS) that handles complex SQL queries and complies with ACID model of databases. PostgreSQL is cross-platform and it can run on many operating systems
The design of the structures, ER schemas and logical schemas of the data warehouse has been realized using the tool Microsoft Visio ®.

4.2 GPDW integrated data model

The project GPDW has been developed to create a data structure to support the integration of genomic and proteomic controlled annotations of different species representing the current biomedical-biomolecular knowledge available on Internet.
GPDW is based on a multi-level data architecture that includes a source-import lower tier, an instance-aggregation middle tier and a concept-integration upper tier. This structure supports integration of data sources that are fast evolving in data content, structure and number, and assures provenance tracking of the integrated data.
The data warehouse uses a flexible data model that provides a unified global view of the data. This model is composed of multiple interconnected modules; it can be extended in a modular way, virtually with no limitations, to include new modules.
Each module represents a single feature that can be a biomolecular entity or a biomedical feature. Additional aspects can extend the basic definition of feature, depending on the information provided by the data source. A feature module can also include similarity and

history aspects [12]. As shown in Figure 8, a feature module, biomolecular entity or biomedical characteristic, can be specified in sub-features, each one with data from one or more sources.

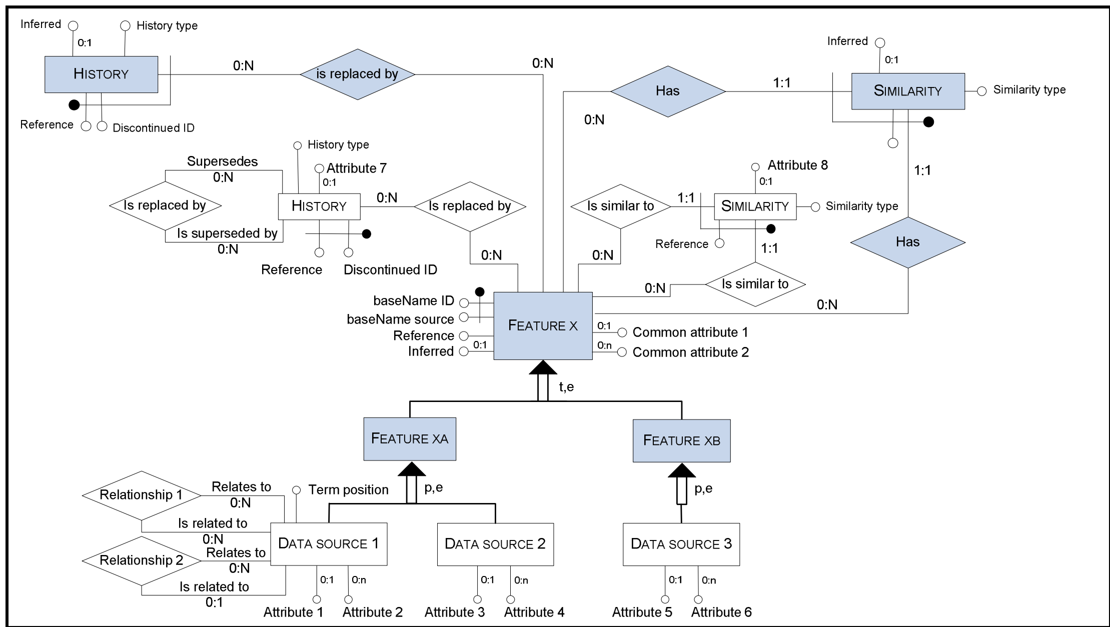


Figure 8 - General conceptual model for features from different sources

The feature module is internally structured in two levels: import level and aggregation level. The import level is represented by separated sub-schemas, one for each data source, that are structured in a global-as-view (GAV) data integration style. In import level, a feature is identified by its id and source attributes, while the *Reference* attribute represents the provenance of the data. In the aggregation level, the main attribute of a feature are associated with a unique OID. Features, both in import and aggregation level, are pair wise associated through association or annotation data. The list of all the feature modules considered by GPDW framework is given in Table 1.

BIOMEDICAL FEATURES	BIOMOLECULAR FEATURES
Biological function feature	Dna sequence
Cdna library	Gene
Clinical synopsis	Protein
Gene expression feature	Small molec
Genetic disorder	Transcript
Enzyme	
Organism	
Pathway	
Protein fam dom	
Publication reference	
Snp	

Table 1 - List of features considered in GPDW

4.3 GPDW framework description

GPDW framework manages the entire process of import and integration, starting from the creation of a supporting database containing metadata, that are needed to run importing and integration procedures.
GPDW follows the Extract-Transform-Load (ETL) paradigm, typical in data warehouse systems and very appropriate when integration of data from multiple applications, typically developed and supported by different organizations, is required [21].
In details, GPDW has multi-level data architecture including:

- Importing level, where biomolecular entities, biomedical features and their annotations and relations data are directly imported from the original sources;
- Aggregation level, where data from different sources regarding the same features are integrated;
- Concept-integration level, where different concept instances of the aggregation tier are integrated and identified by a unique concept OID (Object Identifier).

The main components of the software architecture for the integration of heterogeneous distributed data are sketched in Figure 9.

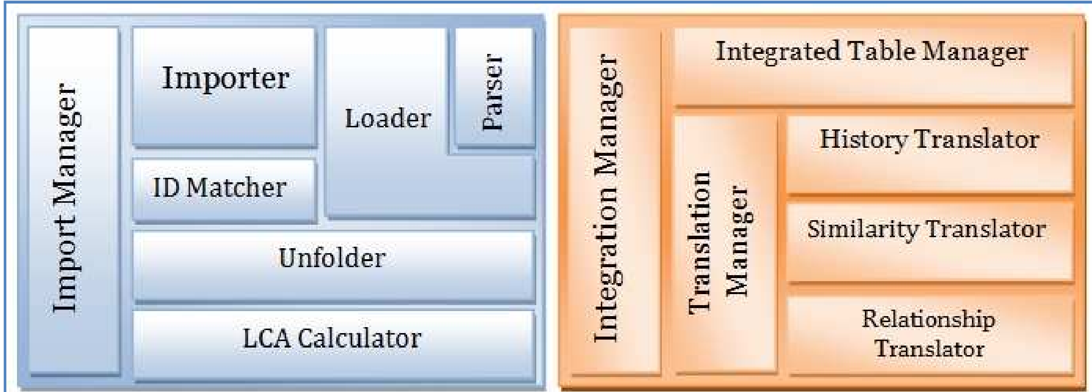


Figure 9 - GPDW software architecture components

The database generated by the execution of this software framework is composed of four schemas:

- *public* schema: it contains imported tables, where there are stored data directly imported from source files, and aggregated level tables related to them, that are generated during the aggregation step.
- *flag* schema: it contains tables that store encoded values of public schema tables.
- *metadata* schema: it stores tables containing metadata, i.e. information about imported sources, imported files, defined features and their properties and relations.

- *log* schema: it contains temporary and supporting tables created by automatic procedures to manage complex operations and specific requirements; this schema also includes tables filled by the records removed from public schema tables, such as duplicated entries or inconsistent values, in order to keep trace of all the imported data and their anomalies.

The whole ETL process can involve considerable complexity and significant operational problems caused by improperly designed processes. In many cases, Extract part is the most challenging aspect of the process, since extracting data correctly sets the stage for how subsequent processes go further. In case of GPDW, both extraction and transformation phases require the implementation of abstract and generic procedures that can be customized to face the heterogeneity of data representation and format, the evolution of data in number and structure [22][23].

4.4 Data import procedures

The importing phase was designed to be flexible and easy extensible. In order to reach this purpose, the whole process is guided by xml configuration files.
Data import operations firstly require the registration of the data source and its feature tables in the configuration files *GPDW_definition.xml*.
GPDW_definition.xml must contain the list of all the data sources considered by the project, together with their high-level characteristics, the features and relations between features that the source treats. Moreover, when the considered source contains data about a feature not yet considered by GPDW, first of all it is necessary to define the new feature in the configuration file *feature_definitions.xml*, specifying if it is a biomedical feature or a biomolecular entity, if ontological and if it provides data about history and similarity.
Secondly, data import procedures require the execution of the standard processes that will be described below. The main software components involved in importing procedures are:

- ImportManager
- Importer
- Loader
- Parser

The main tasks performed by the automatic procedure for data import and the interactions between the components are illustrated in Figure 10.

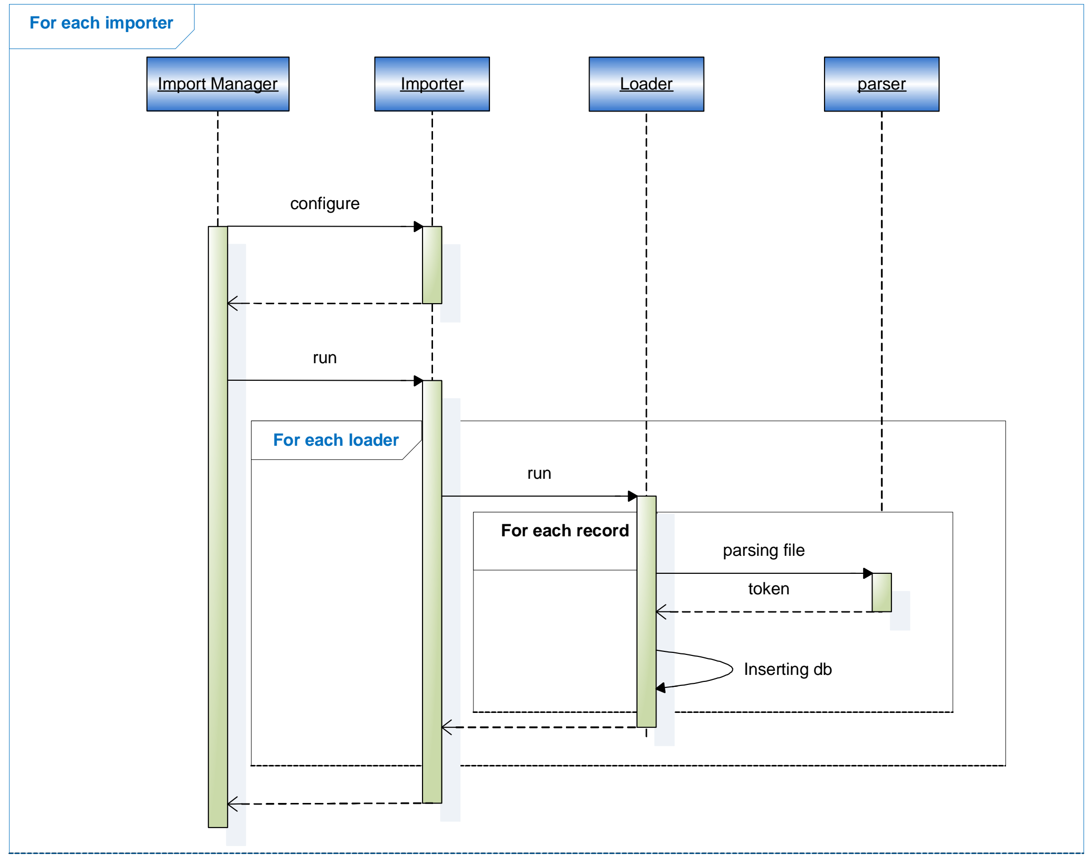


Figure 10 - Sequence diagram of import process

ImportManager.java is the singleton class that guides the whole importing layer. This class instantiates, configures and executes the objects *GenericImporter.java* and *DuplicatesChecker.java*. The former manages the operations to configure and start the import phase of each single source; the latter calls the methods to set constraints and to create indexes on the created importing level database's tables.
The importing phase is concluded by a standard unfolding process of the DAG ontology described by the imported data; an instance of class *UnfolderHelper* manages all the operations to calculate the position of each feature record and the level of the nodes in the ontology. For each couple of instances of a given feature, the Lowest Common Ancestor (LCA) is calculated to complete the analysis of the ontological data structure.

For each data source that is going to be imported, *GenericImporter.java* checks the list of source files and the names of Loader classes. This class can be directly used to instantiate, configure and run the Loaders used for the import of each source file or it can be extended by specific source Importer.
Source Importer can override the inherited methods and, in addition, it may include pre-processing operations and particular functions to manage non-standard properties of the considered data source.

The abstract class *GenericLoader.java*, which is extended by all specific file Loaders, implements the standard operations that actually import data in the data warehouse:

- It reads the part of *GPDW_definition.xml* referred to the considered file in order to get information about the presence of external reference, relationship, history, similarity and association relations between features;
- For each type of relation, it instantiates the specific relation DataLoader, where the database's tables will be created and populated;
- For each relation, it configures lists of specific entries and additional parameters that will contain data to store in the DW. The lists are inserted into HashMap structures using as key the unique handle defined for each relation in the configuration file;
- When file parsing is done, it processes lists of entries and it executes operations to populate database tables through specific DataLoaders;
- In the end, it executes *commit()* of the updates, it generates statistics information to insert in metadata schema and it closes the open connections to the database. In case of error, *rollback()* is executed.

The specific relation DataLoaders implemented in GPDW framework are: *SimilarityDataLoader*, *AssociationDataLoader*, *RelationshipDataLoader*, *HistoryDataLoader* and *ExternalReferenceDataLoader*. All of them have common tasks:

- Creation of the imported level tables;
- Definition of the flags associated to the fields that will be codified;
- Checking of identifiers, provided by the source files, through an instance of the class *IdentifierMatcher.java*. Identifiers must conform to the regular expressions defined in PCRE format [24] in the configuration file;
- Automatic insertion of the entries into the created table of the data warehouse.

The specific file Loader extends an object of type Parser. The distinction between Parser and Loader is not well defined for many data sources.
Usually, a parser extracts data from input source files and produces data tokens usable by the loader, that is responsible for associating a semantic meaning to the tokens and inserting them into the database.
However, in many cases, loader needs to analyze tokens received from the parser in details, by executing some parsing operations itself. GPDW framework includes standard and parsers for the most common file formats:

- *FlatFileParser*: it is used to manage generic flat files without header. This class is extended by *FlatFileWithHeaderParser* and *FlatFileWithFixLengthHeaderParser* classes, which manage flat files that include header lines;

- *TabularFileParser*: it is used to manage tabular files that use single field separator. This class is extended by other two classes, *TabularFileWithHeaderParser* and *TabularFileWithFixLengthHeaderParser*, which manage tabular files that include header lines;
- *TabularFileMultipleSeparatorParser*: it is used to manage tabular files with multiple field separator. This class is extended by *TabularFileWithHeaderMultipleSeparatorParser* which manage tabular files with multiple field separator that include header lines.

Loader extending *FlatFileParser* implements and overrides its abstract method *onNextRecord()*, that handles the processing of tokens extracted from data file; loader extending *TabularFileParser* overrides its abstract method *onNextLine()*, that handles the processing of single line extracted from the file.

The description of the framework reveals that it shows many limitations because there are some elements that are not sufficiently generalized.
For example, the "entry" objects are specific for each type of data, even if they represent the same concept in the database language. Furthermore, the management of source level tables, containing data about feature's characteristics as they are provided by the source file, is not enough generalized because it is totally left to the specific file Loader, that it is responsible for creation and population of these tables.
As a consequence, the project is plenty of useless repetitions of code and of specific classes that do not follow the same procedural rules. It means that any possible modification of the framework will require the redefinition of all the classes previously implemented.
As mentioned in the previous pages, the software platform includes a module named *DuplicatesChecker.java* that is dedicated to the elimination of duplicated tuples and the aggregation of those data referred to the same feature but stored in different records of the same table [25]. DuplicatesChecker is executed at the end of the import process, before enabling primary and foreign keys, creating indexes and setting constraints on tables' fields in order to point out inconsistencies and to improve query response time.
The current software architecture, on the contrary, does not include any module to perform additional operations that could be necessary to complete the import phase in the correct way for those data sources that do not fully fit the automatic general procedures.

4.5 Data integration procedures

The main operations performed for data integration are described in the sequence diagram in Figure 11. These tasks can be grouped in two phases: aggregation and integration. In the former, data from the different sources, imported in the previous data import step, are gathered and normalized into a single representation in the aggregation level. In the latter,

data are organized into informative clusters in the concept-integration tier of the global model [2][12].

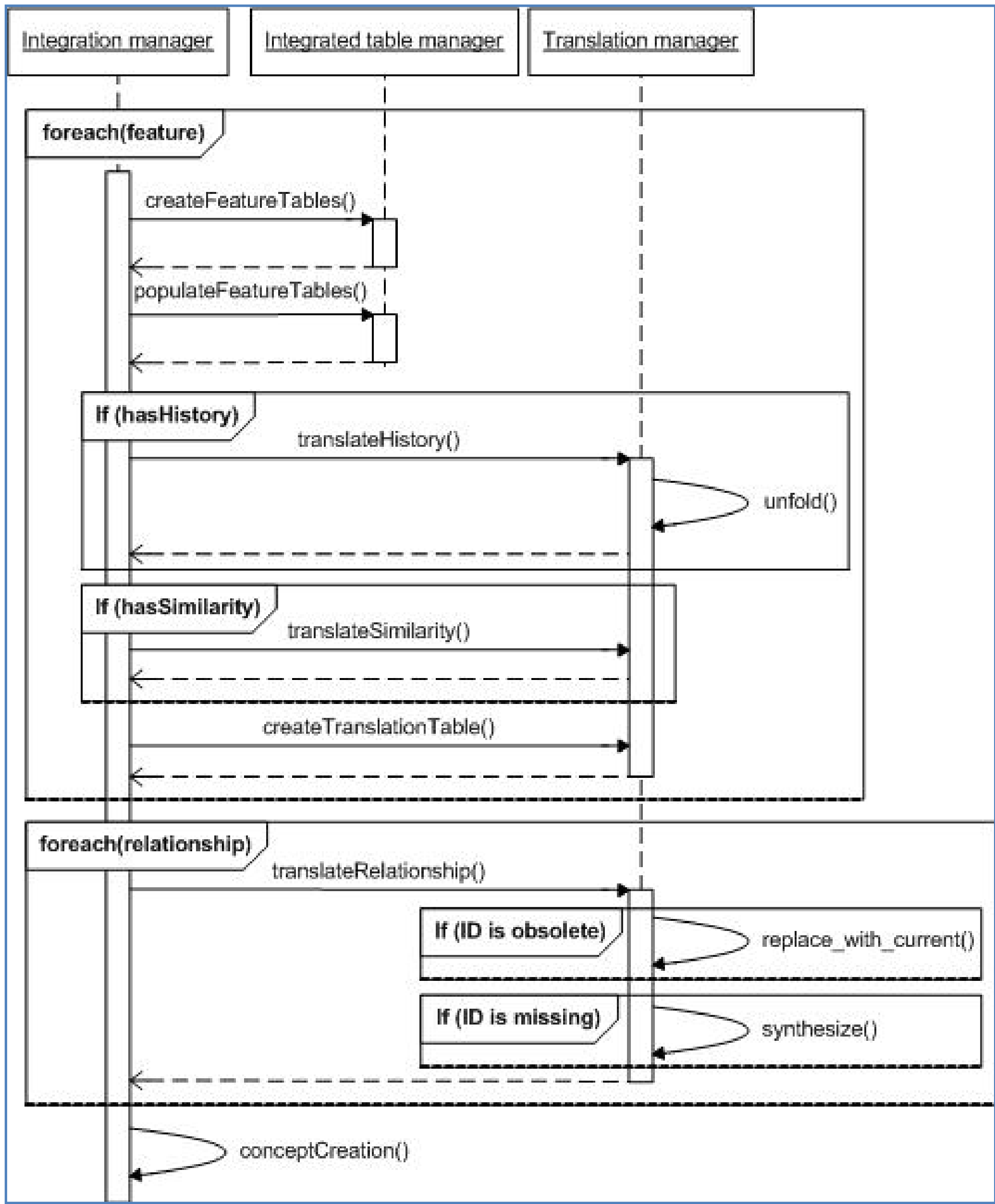


Figure 11 - Sequence diagram of integration process

During the initial aggregation phase, integrated tables are created and populated starting from imported data.

Then, similarity integrated tables, containing data about similarity between different entries of the same feature (e.g. homology between genes or proteins) and historical integrated tables, containing obsolete feature identifiers and the IDs to which they have been propagated, are created by translating the IDs provided by the data sources to data warehouse OIDs.

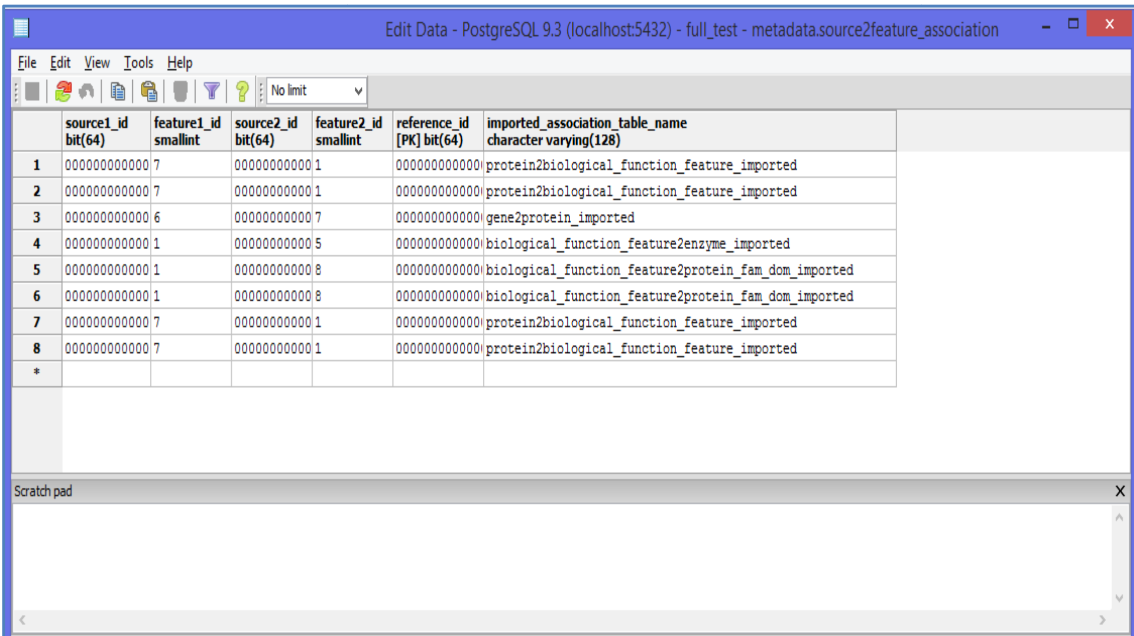
Translation tables for biomolecular entity and biomedical feature IDs are also created by using translated similarity data and unfolded historical ID data. These tables are used as main entry points to navigate the data warehouse. Finally, associations (annotations) between pairs of feature are created by performing OID translation of the imported association data expressed through the source IDs. Association data may refer to features that have not been imported in the data warehouse. In this case, missing integrated feature entries are synthesized and marked as such.
During the final integration phase, by doing a similarity analysis, it is tested whether single feature instances from different sources represent the same feature concept. In positive case, they are associated with a new single concept OID.
At the end of the integration process, on all integrated tables indexes, primary and foreign keys and integrity constraints are enabled in order to detect possible data duplications and to improve the time of access to the integrated data.

4.6 Metadata computation and storage

The *metadata* schema is created to store the information describing the data in the *public* schema of the database. In details, it contains 29 tables used to store framework metadata, such as general information about the data stored by imported and integrated tables.
These tables are created at the moment of the launch of the program, before the beginning of importing procedures, by three different classes:

- *DatabaseMetadata.java* reads the file *db_config.xml* and it creates *metadata.database* and *metadata.creation_step* tables, containing broad information about the created database and each performed step at each level of the global model;
- *FeatureMetadata.java* reads the file *feature_definition.xml* and it stores the metadata of all the features defined in that configuration file. For example, *metadata.feature_association* table contains the list of all the annotations, providing annotation complete name with encoded values of the affected features;
- *ReferenceDB.java* reads the file *GPDW_definition.xml* and generates metadata such as name of main tables, which sources are imported and which are synthesized, which sources/features are ontological, regular expression patterns and many other information.

An example of table in metadata schema is displayed in Figure 12, where the table *metadata.source2feature_association* contains the records related to the import of GOA database. The picture shows that features identifier values are encoded using small integers and that database identifiers are provided as bitmap codes.



	source1_id bit(64)	feature1_id smallint	source2_id bit(64)	feature2_id smallint	reference_id [PK] bit(64)	imported_association_table_name character varying(128)
1	000000000000	7	00000000000	1	000000000000	protein2biological_function_feature_imported
2	000000000000	7	00000000000	1	000000000000	protein2biological_function_feature_imported
3	000000000000	6	00000000000	7	000000000000	gene2protein_imported
4	000000000000	1	00000000000	5	000000000000	biological_function_feature2enzyme_imported
5	000000000000	1	00000000000	8	000000000000	biological_function_feature2protein_fam_dom_imported
6	000000000000	1	00000000000	8	000000000000	biological_function_feature2protein_fam_dom_imported
7	000000000000	7	00000000000	1	000000000000	protein2biological_function_feature_imported
8	000000000000	7	00000000000	1	000000000000	protein2biological_function_feature_imported
*						

Figure 12 – metadata.source2feature_associaton database table

After an exhaustive analysis of the procedures dedicated to the metadata creation and the evaluation of metadata tables' content, the reengineering work proceeded with the optimization of metadata, by adding the information still ignored or not properly generated. Indeed, the correct computation of metadata is fundamental to check the correspondence between what defined in conceptual and logical diagrams, on the basis of the global data schema, and what created in the database through the implementation of the set of automatic procedures.

5. GPDW framework enhancements

The description of the framework architecture provided in the previous chapter highlights design limitations and the necessity for a complete generalization of importing procedures. This Thesis is mainly developed on the import process of GPDW framework, by providing a higher abstraction of general procedures and by extending those operations that show some limitations in design and implementation.

In following sections, there will be described the details of the analysis of procedures already realized and new implementation choices, based on abstraction and customization aspects. The result of this renovation work is sketched in Figure 13, where new modules for post-processing and data recovery operations are integrated at different level of the architecture with already existing components to build the final structure of the framework.

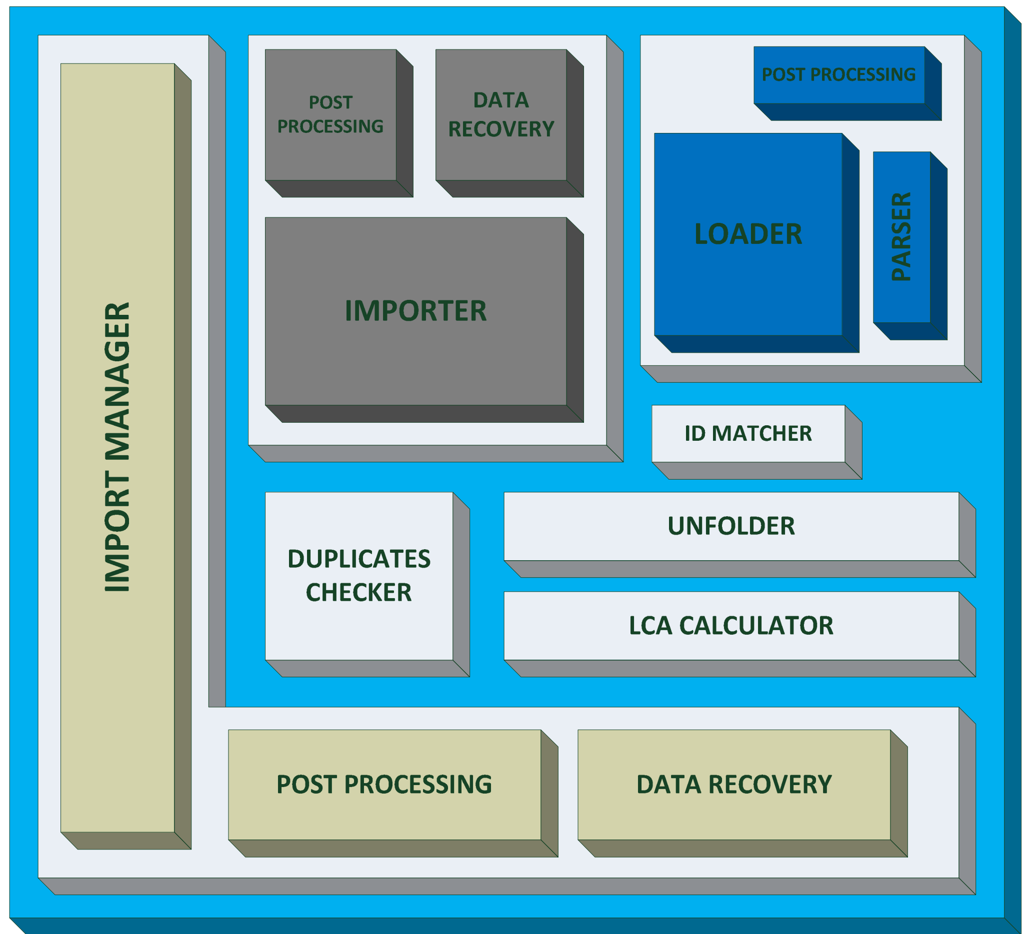


Figure 13 - GPDW framework import layer components

5.1 Enhancement of *GPDW_definition.xml* configuration file

The entire import process is guided by xml configuration files in order to create a structured definition of the framework content and to reduce the amount of Java code implementing the automatic import procedures.
The file *GPDW_definition.xml* includes the description of all the data sources considered by the project, both imported and synthesized. Each imported data source definition must contain all the necessary information to correctly understand data from its files. In details, for each source the following elements are defined:

- <*importer*> (required): it can be the GenericImporter class itself or a specific source Importer class, in case of additional operations to the standard ones are necessary;
- <*post_processing*> (optional): the list of the names of classes that handles the post-processing operations at different levels of the import layer.
- <*data_recoveries*> (optional): it contains the names of classes dedicated to the retrieval of data from *log* schema in different phases of the import procedure;
- <*file_definition_list*> (required): the list of considered source files, grouped into documentation files and data files;
- <*feature_list* > (optional): the list of features to which imported data refer;
- <*feature_association_list*> (optional): the list of annotations to which imported data refer.

In the file list, every <*data_file_definition*> element must specify in its attribute *loader* the name of the Loader object that handles the population of database entries, according to the information provided by the file.
Element <*file*> is univocally identified by its attribute *handle*; it contains information about the URL address for the download of the file from the source website. This element also allows the possibility of specifying a list of additional parameters, which are passed to the Loader in the moment of configuration, that contains useful information not explicitly given by the data in the file.
For each file it is necessary to define which features its records describe and which type of information it provides about these features (sub-features, external reference, relationship, history and similarity). Moreover, the file defines which annotations are described by its data, including possible relations with data in other sources.
All these information must be reflected by what is defined later in the detailed description of <*feature_list*> and <*feature_association_list*>.
Figure 14 shows the XML definition of the considered files of OMIM data source.

```
<data_source_definition handle="omim" name="omim" import="false" download="true">
  <description>Description of omim source</description>
  <importer>it.polimi.gfinder2.importer.omim.OmimImporter</importer>
  <download_uri>
    <file_definition_list>
      <data_file_definition loader="it.polimi.gfinder2.importer.omim.OmimLoader">
        <url handle="omim_omimData_DownUrl" name="ftp://.../omim.txt" download="true">
           <file handle="omim_omimData_File" import="true" >
              <file_path>omim/omim.txt</file_path>
              <loader_post_processing>it.pol...omim.OmimLoaderPostProc</loader_post_processing>
           </file>
        </url>
        <data>
         <feature_list>
           <feature handle="gene">
             <relationship handle="omim_gene_relationship" />
             <history handle="gene_history" />
           </feature>
           <feature handle="genetic_disorder">
             <relationship handle="omim_disorder_relationship" />
             <history handle="genetic_disorder_history" />
           </feature>
           <feature handle="clinical_synopsis">
             <relationship handle="omim_clinical_synopsis_relationship" />
           </feature>
         </feature_list>
         <feature_association_list>
           <feature_association handle="gene2genetic_disorder">
              <match handle="omim2omim" />
           </feature_association>
           <feature_association handle="gene2clinical_synopsis">
              <match handle="omim2omim" />
           </feature_association>
           <feature_association handle="genetic_disorder2clinical_synopsis">
              <match handle="omim2omim" />
           </feature_association>
         </feature_association_list>
        </data>
      </data_file_definition>
      ......
      <data_file_definition loader="it.polimi.gfinder2.importer.omim.PubmedLoader">
        <url handle="omim_pubmed_cited_DownUrl" name="ftp://.../pubmed_cited" download="true">
           <file handle="omim_pubmed_cited_File" import="true">
              <file_path>omim/pubmed_cited</file_path>
           </file>
        </url>
        <data>
          <feature_association_list>
             <feature_association handle="gene2publication_reference">
                <match handle="omim2pubmed" />
             </feature_association>
             <feature_association handle="genetic_disorder2publication_reference">
                <match handle="omim2pubmed" />
             </feature_association>
          </feature_association_list>
        </data>
      </data_file_definition>
    </file_definition_list>
  </download_uri>
  ......
```

Figure 14 - Example of source file definition

Following the definition of source files, there is the definition of features considered by such files that are, in case of OMIM, gene, genetic disorder and clinical synopsis.
In order to manage both feature and sub-feature entities in the same way, the existing feature definition was generalized by creating a new template called *base_feature_definition*. So, it is possible to treat both features and sub-features as the same object type from the computational point of view, avoiding to create ad-hoc procedures to manage sub-feature data as it was in the previous version of software architecture.

This base definition embodies the essential elements shared by main features and sub-features, i.e. source tables, hierarchical relationship, history and similarity. It can be also extended to include additional elements, i.e. external references, regular expressions and URI (Uniform Resource Identifier) addresses, which are considered meaningful only in case of parent features.
The XML Schema code that shows this enhancement is given in Figure 15.

```
.....
<xs:complexType name="base_feature_definition">
   <xs:sequence>
      <xs:element name="source_table" type="source_table_definition"
                    minOccurs="0" maxOccurs="1" />
      <xs:element name="relationship" type="feature_relationship_type"
                    minOccurs="0" maxOccurs="1" />
      <xs:element name="history" type="feature_history_type" minOccurs="0"
                    maxOccurs="1" />
      <xs:element name="similarity" type="feature_similarity_type"
                  minOccurs="0" maxOccurs="1" />
   </xs:sequence>
   <xs:attribute name="name" type="xs:string" use="required" />
   <xs:attribute name="handle" type="xs:string" use="required" />
</xs:complexType>
.....
<xs:complexType name="data_source_feature">
   <xs:complexContent>
      <xs:extension base="base_feature_definition">
         <xs:sequence>
           <xs:element name="external_reference" type="feature_ext_ref"
                          minOccurs="0" maxOccurs="1" />
           <xs:element name="sub_feature_list" minOccurs="0"
                          maxOccurs="1">
           <xs:complexType>
            <xs:sequence>
              <xs:element name="sub_feature" type="base_feature_definition"
                            minOccurs="1" maxOccurs="unbounded" />
            </xs:sequence>
           </xs:complexType>
           </xs:element>
           <xs:element name="reg_expr_list"
                         type="data_source_feature_reg_expr"
                         minOccurs="0" maxOccurs="1">
           </xs:element>
           <xs:element name="display_uri" type="text_url"
                         minOccurs="0" maxOccurs="1" >
                 <xs:unique name="feature_display_uri_unique_key">
                    <xs:selector xpath="d:url" />
                     <xs:field xpath="@display_class" />
                 </xs:unique>
           </xs:element>
         </xs:sequence>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
.....
```

Figure 15 – XML Schema description of feature definition

Finally, associations between the features described by the considered source and other features, given by the other databanks of the project, are described in order to reflect those associations included in the definition of the source files.
The association data provided by Gene Ontology are displayed in Figure 16.

```
  .....
<feature_association_list>
   <feature_association handle="biological_function_feature2enzyme"
           source_feature_handle="biological_function_feature"
           destination_feature_handle="enzyme" >
       <association_type handle="related_to" />
       <match handle="go2expasy_enzyme">
          <from_data_source handle="go" />
          <to_data_source handle="expasy_enzyme" />
       </match>
     </feature_association>
     <feature_association handle="biological_function_feature2pathway"
           source_feature_handle="biological_function_feature"
           destination_feature_handle="pathway" >
       <association_type handle="related_to" />
       <match handle="go2reactome">
          <from_data_source handle="go" />
         <to_data_source handle="reactome" />
       </match>
     </feature_association>
  </feature_association_list>
</data_source_definition>
```

Figure 16 - Example of feature association definition

5.2 Enhancement of *feature_definitions.xml* configuration file

The file *feature_definitions.xml* includes the description of all the features considered by the project, already listed in Table 1.

For each feature, it is specified as either biomedical feature or biomolecular feature, if it is ontological feature (it has hierarchical relationships), if it has history and similarity relations. All the feature elements contain the definition of the aggregation level tables, where eventual additional attributes could extend the standard template used as skeleton for the database tables.

In the same manner, the structure of history, similarity and association tables is defined using the templates specified at the top of the configuration file. As an example, template used for similarity tables is shown in Figure 17.

```
<template name="feature_similarity">
   <table name="baseName_similarity" xmlns="http://polimi.it/gfinder2/table_definition">
     <attribute name="baseName_oid" type="BIGINT" nullable="false"/>
     <attribute name="similar_baseName_oid" type="BIGINT" nullable="false"/>
     <attribute name="reference" type="DATA_SOURCE_ID_TYPE" nullable="false"/>
     <attribute name="similarity_type" type="SMALLINT" nullable="false" encoded="true"/>
     <attribute name="inferred" type="INTEGER" nullable="true" encoded="true"/>
     <primary_key>
         <attribute name="baseName_oid" />
         <attribute name="similar_baseName_oid" />
         <attribute name="reference" />
     </primary_key>
     <foreign_key name="baseName_term_fk" references="baseName">
         <attribute name="baseName_oid" references="baseName_oid" />
     </foreign_key>
     <foreign_key name="baseName_simil_term_fk" references="baseName">
         <attribute name="similar_baseName_oid" references="baseName_oid" />
     </foreign_key>
     <index name="baseName_similarity_idx1">
         <attribute name="baseName_oid" integrated="true" />
     </index>
     <index name="baseName_similarity_idx2">
         <attribute name="similar_baseName_oid" integrated="true" />
     </index>
     <index name="baseName_similarity_idx3">
         <attribute name="similarity_type" integrated="true" />
     </index>
     <index name="baseName_similarity_idx4">
         <attribute name="inferred" integrated="true" />
     </index>
   </table>
</template>
```

Figure 17 - Similarity integrated table template

These templates have been designed as general as possible and contain only the fields shared among the table of same type but of different features. Instead, there is no template designed for the feature tables of the imported level. Thus, all the imported feature tables must be described by the specific source file Loader. This is clearly a limitation that produces code redundancy and neglects the possibility to extend generalized automatic procedures to import the feature tables.

A detailed analysis of existing database schemas and importing procedures brings the work in the direction of designing a new part of xml code, and the related xsd code to validate it. This new block, represented by the element <custom_source_tables>, contains the description of imported level feature tables; each source table definition extends one of the following novel table templates, that have been added to the existing templates:

- *feature_source_imported*, used for the main source feature table;
- *feature_relationship_imported_temp*, used for relationship imported table;
- *feature_additional_table*, for integrated feature table with primary key;
- *feature_additional_table_unique*, for integrated feature table with unique index;
- *fetaure_additional_table_imported*, for imported feature table with primary key;
- *fetaure_additional_table_imported_unique*, for imported feature table with unique index.

Figure 18 illustrates the definition of all the source tables created by the automatic importing procedures for the source ExPASy ENZYME. The standard templates for additonal source feature tables are shown in Figure 19.

```
<custom_source_tables>
 <source name="expasy_enzyme">
  <table_list>
    <table_derived name="baseName" baseTemplate="feature_source_imported" xmlns="...">
         <attribute name="name" type="VARCHAR(256)" nullable="false" />
         <attribute name="catalytic_activity" type="VARCHAR" nullable="true" />
         <attribute name="cofactor" type="INTEGER" nullable="true" encoded="true" />
    </table_derived>
    <table_derived name="baseName_comment" baseTemplate="feature_additional..." xmlns="...">
          <attribute name="comment" type="VARCHAR" nullable="false" />
          <primary_key>
             <attribute name="comment" />
          </primary_key>
    </table_derived>
    <table_derived name="baseName_alternative_name" baseTemplate="feature_addit..." xmlns="...">
          <attribute name="alternative_name" type="VARCHAR(256)" nullable="false" />
          <primary_key>
             <attribute name="alternative_name" />
          </primary_key>
    </table_derived>
    <table_derived name="baseName_action" baseTemplate="feature_additional..." xmlns="...">
          <attribute name="action" type="SMALLINT" nullable="false" encoded="true" />
          <primary_key>
             <attribute name="action" />
          </primary_key>
    </table_derived>
   </table_list>
 </source>
 .....
```

Figure 18 - Source tables definition of ExPASy ENZYME databank

```
<template name="feature_additional_tables_imported">
  <table name="baseName_tableName" xmlns="http://polimi.it/gfinder2/...">
    <attribute name="baseName_oid" type="BIGINT" nullable="false" />
    <attribute name="reference_file" type="bit(256)" nullable="false" />
    <primary_key>
       <attribute name="baseName_oid" />
    </primary_key>
    <foreign_key name="baseName_tableName_fk" references="baseName">
       <attribute name="baseName_oid" references="baseName_oid" />
    </foreign_key>
    <index name="baseName_tableName_baseName_oid_idx">
       <attribute name="baseName_oid" />
    </index>
  </table>
</template>

<template name="feature_additional_tables_unique_imported">
  <table name="baseName_tableName" xmlns="http://polimi.it/gfinder2/...">
    <attribute name="baseName_oid" type="BIGINT" nullable="false" />
    <attribute name="reference_file" type="bit(256)" nullable="false" />
    <foreign_key name="baseName_tableName_fk" references="baseName">
       <attribute name="baseName_oid" references="baseName_oid" />
    </foreign_key>
    <index name="baseName_tableName_baseName_oid_idx">
       <attribute name="baseName_oid" />
    </index>
    <unique name="baseName_tableName_uidx">
        <attribute name="baseName_oid" />
    </unique>
  </table>
</template>
```

Figure 19 - Templates for imported additional source tables

5.3 Automatic import of source tables

In this part of the Thesis, there are illustrated the components designed to extend automatic procedures of importing process to the creation of database source tables.
The java classes representing these components have been designed by using the same conceptual organization of those classes, already present in the previous version of GPDW framework, that manage the importing operations for association, similarity and history tables. The class *RelationshipDataLoader.java*, on the other hand, was completely rebuilt in order to fit new updates of the proposed solution that ensures higher accuracy of imported data and reduces the possibility of unexpected errors.
It is the GenericLoader class, as done for the specific relation DataLoader, that instantiates the objects of new classes, and that configures, and finally processes, the lists of specific entries and additional parameters that will contain data from source files.
The new workflow diagram of GenericLoader.java is illustrated in figures 20 and 21.

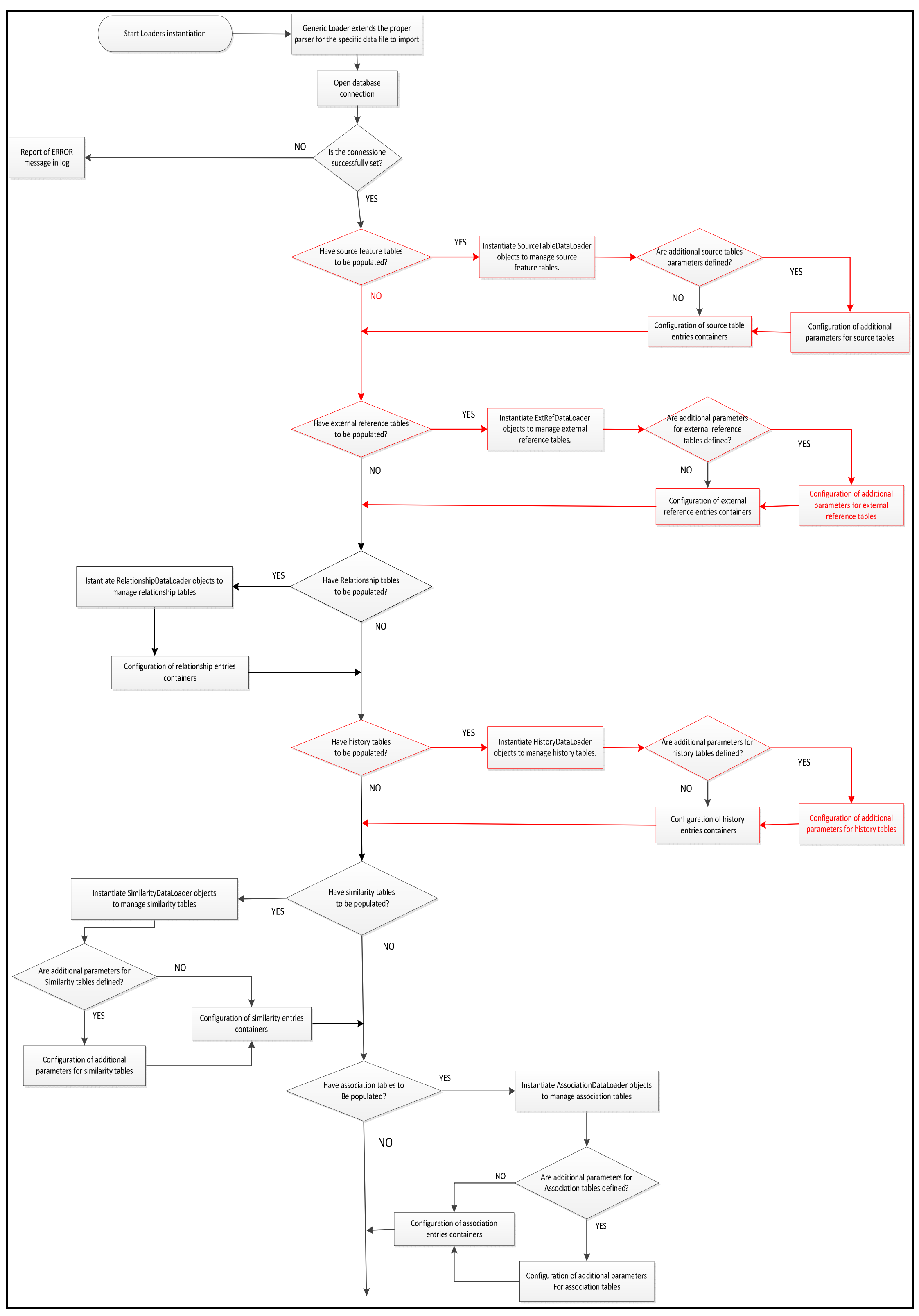


Figure 20 - GenericLoader workflow - part I

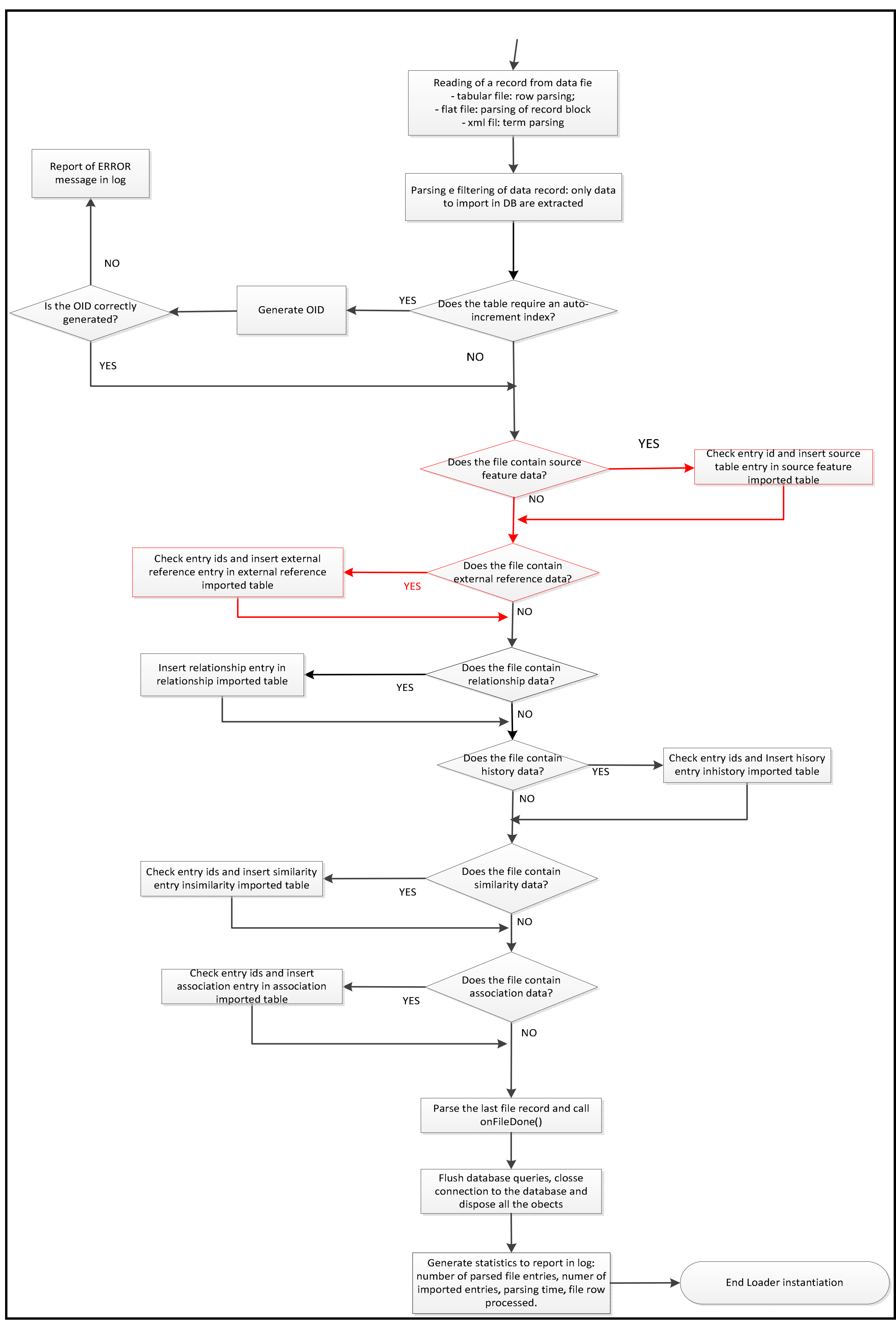


Figure 21 - GenericLoader workflow - part II

5.3.1 Generic source tables Loader

The class *MainSourceTableDataLoader.java* handles the generic logic to configure the import process of main source tables, also called source feature table. This component performs many standard operations that can be summarized in four steps:

- Configuration: an instance of the class is created, a connection to the DB is opened, the database table is created, if it does not already exist, flags are defined for encoded fields, if present. Besides, an instance of *IdentifierMatcher* class is created to recover the regular expressions defined in *GPDW_definition.xml* for the considered feature;
- Insertion of entry: for each entry, before inserting all the information into the database, the source identifier is matched to the regular expressions list. In case of failure, an error message is displayed in the log and the entry is discarded. Otherwise, a new unique OID is generated for the entry, incrementing the relative counter, and the entry is inserted in the TableLoader instance attached to the supplied database connection. *TableLoader.java* is a class that is a helper used to fill the database tables hiding all the low level SQL details.
- Insertion of additional entry: in case of additional source tables, which are related to the main one by a foreign key constraint, the method to insert additional entries in the proper tables are called passing the unique OID as parameter. The operations for inserting additional entries are fully managed by the class *AdditonalSourceTableDataLoader.java* (see section 5.3.2);
- Disposing: flags and TableLoader instance are flushed so that the commit of all the database operations is allowed. All the connections to the database are closed and statistical information are generated and inserted in *metadata* schema.

The creation of the table in the public schema of the DB is done by using the information passed as parameters by GenericLoader. These parameters include all the required information defined in the configuration files:

- *factory* – the ConnectionPoolFactory instance to access the database;
- *sourceHandle* – the unique handle of the data source;
- *featHandle* – the unique handle of the feature;
- *featName* – the name of the feature;
- *mainFeatHandle* – handle of parent feature, if present;
- *params* – list of attributes derived from <*custom_source_tables*>.

The management of encoded value is done in the configuration method by creating a list of flags implemented through a HashMap structure, where column's name is used as key: The value of the codified fields is handled by the interface *IFlag*, specially created for this task.

5.3.2 Generic additional source tables Loader

The class *AdditionalSourceTableDataLoader.java* handles the generic logic to configure the import process of additional source tables.
Additional source tables are used by GPDW architecture to expand the information about a feature (or sub-feature); they include some relevant fields extracted from source files in order to store them in more compact tables that can be quickly accessed and queried. Additional tables cannot exist without the main feature table they refer to.
This component is closely related to the component MainSourceTableDataLoader, reflecting the connection that exists between the tables they create.
Indeed, AdditionalSourceTableDataLoader performs the same basic operations in the configuration step that are listed below:

- Open a connection to the database;
- Creates the database table in *public* schema;
- Create an instance of TableLoader and attach it to the open DB connection;
- Define flags for encoded fields.

Then, it works by inserting the entries one by one if no exceptions have been raised. In case of number or type of optional parameters does not match the definition of the associated table, that is derived by merging the base template to the attribute list provided in *feature_definitions.xml*, an error message is shown in the log and the entry is discarded. For example, as shown in Figure 18, data imported from ExPASY ENZYME are organized into a single feature table, *expasy_enzyme*, and three additional tables where storing information about enzymes' alternative names, actions and comments provided by databank's curators.

The prior software framework allowed to any Loader of managing the tables they fill, independently from general procedures, via local implementation of insertion operations. Now, the implementation of these two modules makes the insertion of feature entry objects in the database table a broad and standard operation that is exclusively performed by instances of these classes in a more consistent and reliable manner.
The general method used by MainSourceTableDataLoader to insert entry's values in the imported level database table is shown in Figure 22. This method is used by the entire set of Loaders that imports feature records from source files.

```java
public boolean insertMainSourceTableEntry(MainSourceTableEntry e,ReferenceFile refFile)
        throws SQLException{

  if(idMatcher.sourceIdMatchIdentifier(e.getSource_id())){
     long feature_oid = oid_generator.getId();

     try {
          loader.beginRow();
          loader.column(FEATURE_OID, feature_oid);
          loader.column(SOURCE_ID, e.getSource_id());
          loader.column(SOURCE_NAME, e.getSource_name());
          loader.column(REFERENCE_FILE, refFile);
          loader.column(FEATURE_TYPE, e.getFeature_type());
          /* Adding optional params */
          if (e.getParams() != null)
            for (int i=0; i < additional_fields_cnt; i++) {
                  AdditionalParam p = e.getParams().get(i);
                  if(p.getValue() != null)
                   if(p.is_encoded())
                        loader.column(EXTRA_FIELDS + i, flagList.get(p.getColumnName()).
                                                         getFlagId(p.getValueToString()).
                                                         getId());
                    else
                        loader.column(EXTRA_FIELDS + i, p.getValue());
                  else
                    loader.column(EXTRA_FIELDS + i, TableLoader.NULL);
                       }
          else {
             return false;
          }

          loader.endRow();
          main_tbl_entries_cnt++;
     } catch (IOException e1) {
          logger.error("Error in insert entry for table " + tableHandle, e1);
          return false;
     }
     insertAdditionalSourceTables(e, feature_oid, refFile);
     return true;
  }
  else {
        logger.error("Error: source_id does not matches with regular expressions");
        return false;
  }
}
```

Figure 22 – Method *insertMainSourceTableEntry*

5.3.3 Relationship tables Loader

Since the insertion of relationship entries is done in parallel to the insertion of main source tables entries, it is possible that, in a certain moment, the related feature described by the relationship data has not yet been read from the file and, consequently, that a unique OID has not yet been created for the related feature entry.

This eventuality leads to the impossibility of inserting relationship entry content directly into the database, at least by using the current table template in Figure 23, where there are stored only the OIDs of related features.

```xml
<template name="feature_relationship_imported">
 <table name="baseName_relationship" xmlns="http://polimi.it...">
   <attribute name="term_oid" type="BIGINT" nullable="false"/>
   <attribute name="related_term_oid" type="BIGINT" nullable="false"/>
   <attribute name="relationship_type" type="INT" nullable="false" encoded="true"/>
   <attribute name="inferred" type="INTEGER" nullable="true" encoded="true" />
   <attribute name="reference_file" type="bit(256)" nullable="false" />
   <primary_key>
       <attribute name="term_oid" />
       <attribute name="related_term_oid" />
       <attribute name="relationship_type" />
   </primary_key>
   <foreign_key name="baseName_rel_fk1" references="baseName">
       <attribute name="term_oid" references="baseName_oid" />
   </foreign_key>
   <foreign_key name="baseName_rel_fk2" references="baseName">
       <attribute name="related_term_oid" references="baseName_oid" />
   </foreign_key>
   <index name="baseName_relationship_term_idx">
       <attribute name="term_oid" />
   </index>
   <index name="baseName_relationship_rel_idx">
       <attribute name="related_term_oid" />
   </index>
   <index name="baseName_relationship_type_idx">
       <attribute name="relationship_type" />
   </index>
   <index name="baseName_relationship_inf_idx">
       <attribute name="inferred" />
   </index>
 </table>
</template>
```

Figure 23 – Relationship imported table template

In order to manage the issue, the previous version of *RelationshipDataLoader.java* class has been modified by creating a temporary table, in the *log* schema, that will contain the relationship entries filled by the file Loader class. The new template designed to handle relationship data stores the feature id and the source id instead of OIDs.
Finally, before disposing object's instance and closing the connection to the database, the log table is joined to the related main source table to recover features' OIDs and insert all the fields into the imported relationship table, where records are permanently stored.
Figure 24 shows an example of a query generated by the application of new RelationshipDataLoader procedure to the data provided by Gene Ontology.

```sql
INSERT INTO go_cellular_component_relationship
SELECT DISTINCT a.go_cellular_component_oid, b.go_cellular_component_oid,
log.relationship_type, log.inferred, log.reference_file
FROM log.go_cellular_component_relationship_temp_1 AS log
JOIN go_cellular_component AS a
ON log.term_id = a.source_id  AND log.term_source = a.source_name
JOIN go_cellular_component AS b
ON log.related_term_id = b.source_id AND log.related_term_source = b.source_name
```

Figure 24 - Example of query generated by RelationshipDataLoader

This extension and renovation work allows to significantly reduce redundant code and to eliminate some ad-hoc packages and classes, specifically created for each data source.
As discussed in section 3, Chapter 4, at the end of import procedure the module named DuplicatesChecker deals with the elimination of duplicated entries and the aggregation of those data referred to the same feature but stored in different records of the same table.
Since source tables had not been yet considered by this module, these activities have been spread to both main source tables and additional source tables through the use of abstract classes *GenericDuplicatesChecker.java* and *GenericEnableConstraints.java* [25].
In the previous version of the framework, such abstract objects were implemented by specific classes, defined at the top of the xml definition of each imported source. There, the name of table fields implicated in indexes, primary keys and integrity constraints were manually set.
It was an evident design limitation because potential changes to the logical schemas of the database would have required the update of all the parameters used to check duplicates and to enable table's constraints. Hence, the extension of DuplicatesChecker procedures to the source tables, by creating adequate data structure to store information about table fields, was consequently implemented; these structures are filled at the beginning of the import procedure, when configuration files are scanned. and their data can be retrieved by the ImportManager before the execution of DuplicatesChecker.
Thanks to this last modification, in the new version of GPDW framework, the importing process of source tables have been aligned to the importing procedure already implemented for history, similarity, external reference and association table. All the operations regarding the import level have been generalized in order to be automatically guided by configuration files.

5.4 Post-processing and data recovery components

The current section illustrates the main features of new modules designed and implemented to complete the import layer of GPDW architecture.
These components have been designed to answer specific data source requirements that can occur during the whole import process. The analysis of data imported, generated through standard procedures of the existing framework, suggests to create two modules for post-processing and data recovery operations, enabling their execution in different moments of the process, according to the specific requirement to fulfill.
The following sub-sections focus on design choices and adopted technological solutions to implement the components that complete the platform architecture. The addition of these modules requires a reengineering process on the workflow of the ImportManager; the result of these enhancements is shown in figures from 25 to 27.

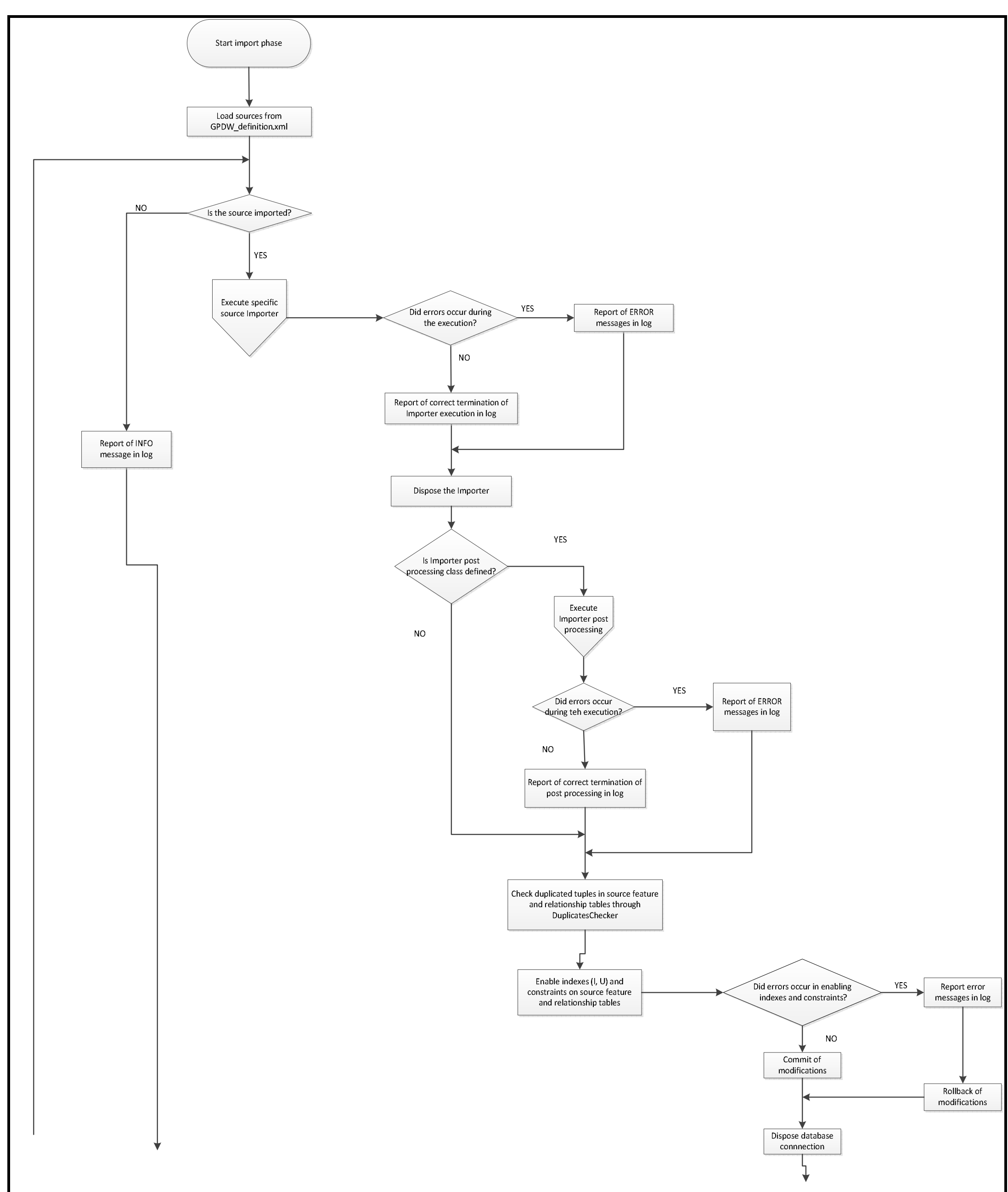


Figure 25 – ImportManager workflow - part I

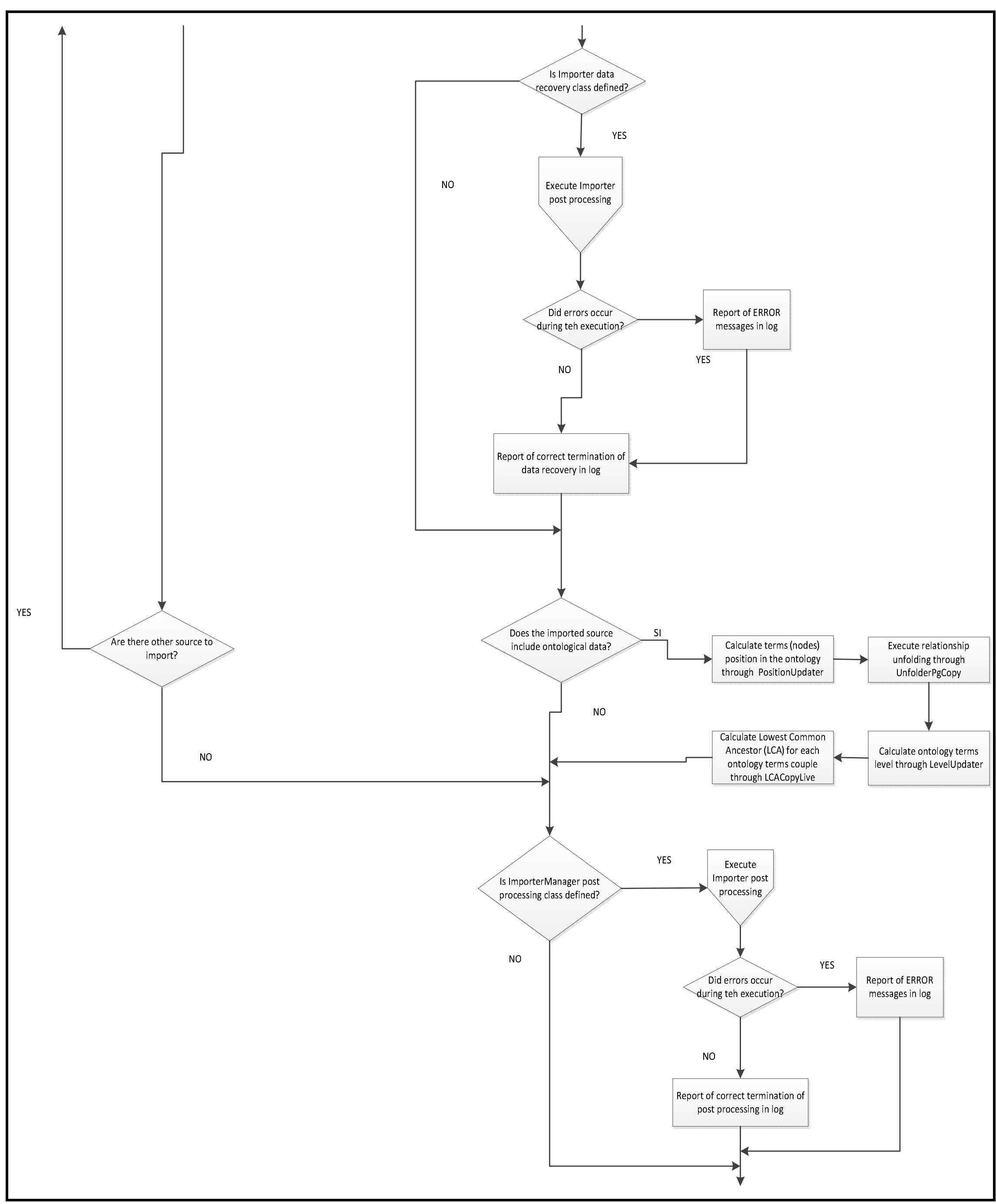


Figure 26 – ImportManager workflow - part II

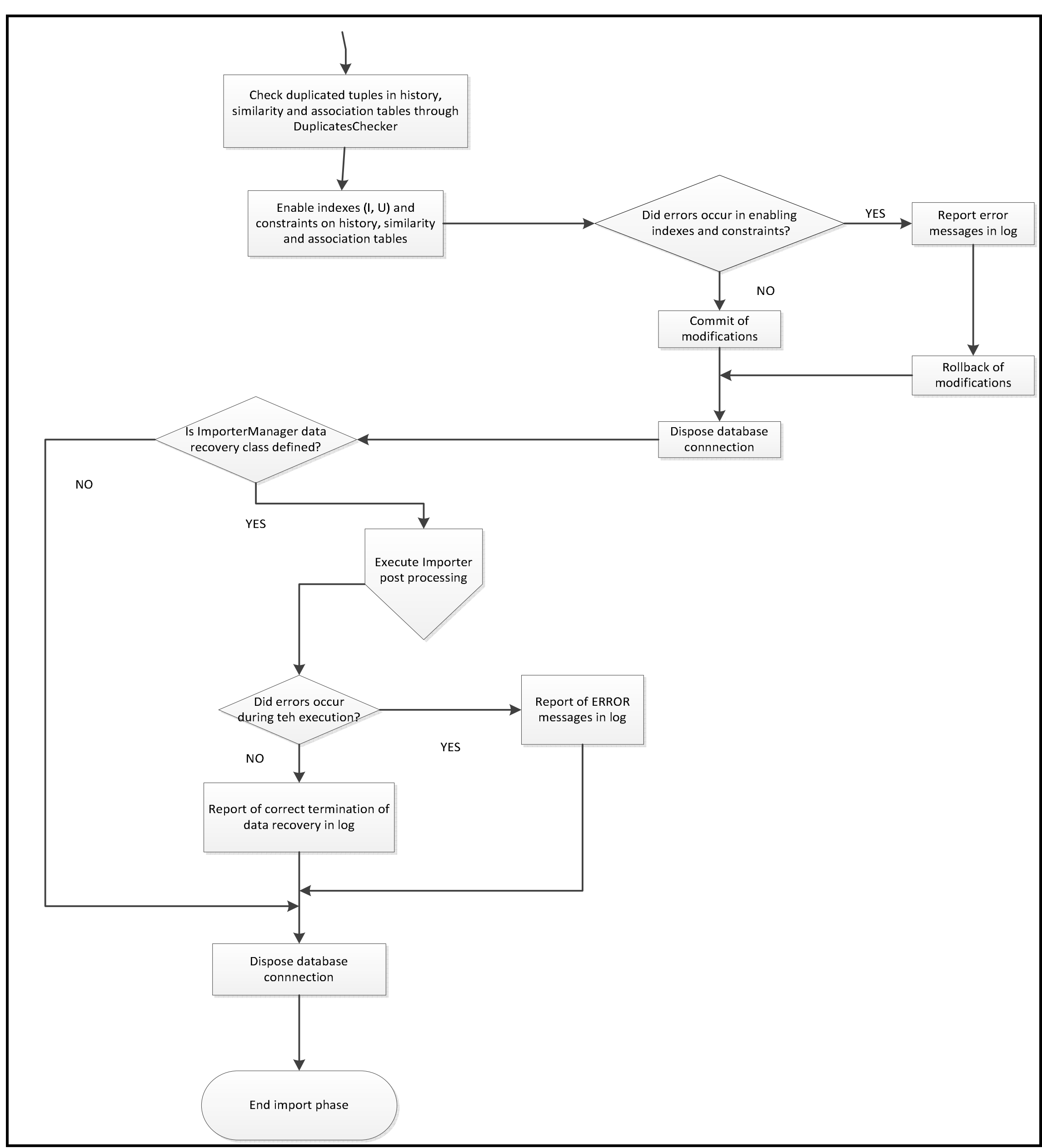


Figure 27 – ImportManager workflow - part III

5.4.1 Post-processing

Post-processing module performs many operations that are specific for each data source that need to instantiate an object of this type. Post-processing operations can be configured and executed at different steps of the import process. In details:

- Loader level: post-processing can regard a single source file, or a group of files using the same Loader class. For example, the specific data file format can require the creation of many tables in *log* schema to temporally store data that cannot be inserted in the proper table at the moment of the parsing. The post-processing classes of this type receive, during the configuration step, the same parameters of the related Loader class to have the possibility of using the same information and extending the set of operations realized by the Loader itself.
- Importer level: additional operations before the execution of DuplicatesChecker procedures on source tables can be useful to merge information from different files of the same data source or to limit the amount of data that will be inserted in error tables created by the duplicates' checking.
- ImportManager level: in many cases it could be necessary that specific Importer class instantiated for a certain source have completed tables population to allows the correct insertion of association, similarity or external reference data from another source. Hence, these post-processing operations are executed before running the DuplicatesChecker on imported relation tables.

Post-processing operations are collected in appositely created classes that must be defined in the *GPDW_definition.xml* configuration file. The name of these classes is usually self-explanatory in order to indicate the main operations performed. All the post-processing classes implement the interface that have been specially created to manage the standard configure, run and dispose operations. These interfaces are:

- *IGenericImportMgrPostProcessing;*
- *IGenericImporterPostProcessing;*
- *IGenericLoaderPostProcessing.*

Specific implementation of post-processing classes will be discussed in Chapter 7, where a use-case scenario will be presented for OMIM data sources.

5.4.2 Data recovery

Data Recovery module is responsible for the retrieval of those information that populate the error tables during the aggregation and elimination process of duplicated tuples. In many cases, it could happen that tables containing entries deleted from database public schema store some useful data. It is verified, for example, when different sources provide feature identifiers that can be matched by the same regular expression and, consequently, the primary key field *source_name* cannot be univocally determined.
Data recovery operations can be configured and executed at different moments of the import process:

- Importer level: when data to recover do not refer to multiple sources, data recovery operations can be executed at the end of source Importer execution, before running Unfolder and Lowest Common Ancestor (LCA) modules.
- ImportManager level: data recovery operations, at this level, represent the last activities of the import process, when necessary.

For example, in case of GAD data source, that is not considered in this Thesis, an ImporterManager data recovery module has been successfully implemented to handle the insertion of data in *gene_similarity_imported* and *genetic_disorder_similarity_imported* according to the value of the source identifiers that are matched to those provided by OMIM files. These operations, before the design of data recovery component, were manually implemented in specific phase of the software execution. As done for post-processing module, data recovery classes are defined in configuration file. They extend an abstract generic class that provides standard methods for configuration, database connection and disposing of objects' instances.

6. Considered data sources

Design and implementation of the database have been realized in three steps:

- Conceptual design: modeling of Entity Relationship (ER) schema that represents informal requirements into a formal description of the considered domain that is independent from the representation model used by the DBMS
- Logical design: translation of relationships and dependencies of conceptual model into a logical structure which can be mapped into the storage objects supported by the DBMS. Like the conceptual schema, logical schema does not depend on physical implementation details.
- Physical design: refinement of the logical schema including the choice of physical parameters to store data, file distribution policy and indexes organization. In an Object database the storage parameters correspond directly to the objects used by the OOP language chosen to implement the applications that will manage and access the data.

The design of ER schemas and logical schemas of GPDW databanks has been realized using the tool Microsoft Visio ®, while the physical implementation is directly connected to the framework procedures previously described. In the next pages, the following information will be point out for each data source considered in this Thesis:

- General description of the source;
- The content of few imported files and the list of tables they populate.
- Conceptual and logical diagrams;

Graphical conventions have been adopted in the design of data source schemas. In ER schemas some attributes are represented in red color to highlight that they are omitted in the importing process. In logical schemas different colors are used for different table types, as explained in Table 2.

Table color	Table type
	ID translation table
	Integrated association table
	Integrated similarity or history table
	Integrated unfolding table
	Integrated relationship table
	Integrated feature table
	Imported association table
	Imported similarity or history table
	Imported unfolding table
	Imported relationship table
	Imported feature table

Table 2 - Table of colors and types of Logical schemas tables

Besides, table's attributes in logical schemas follow a set of rules:

- Fields in bold typeface are mandatory (cannot have null value);
- **PK** means that the field is part of the primary key of the table;
- **FK** means that the field is a foreign key used to link data from two tables;
- **U***n* means that the field is part of a unique index;
- **I***n* means that the field is part of an index;
- Fields preceded by ***** character are encoded and their original values are stored in DB *flag* schema;
- Fields precede by **+** character are encoded and their original values are stored in DB *metadata* schema.

6.1 Gene Ontology - GO

The Gene Ontology (GO) project is a collaborative effort to address the need for consistent descriptions of gene products in different databases. The GO Consortium was established in 1998 as a cooperation between three model organism databanks, *FlyBase* (Drosophila), the *Saccharomyces Genome Database* (SGD) and the *Mouse Genome Database* (MGD). Since then, the GO project has grown including many repositories for plant, animal and microbial genomes [26][27].
The Gene Ontology project has developed three structured controlled vocabularies that describe gene products in term of their associated biological processes, cellular components

and molecular functions. A biological process is a set of molecular events pertinent to the living entity's functionalities. A cellular component is a part of a cell or its extracellular environment, that may be an anatomical structure or a gene product group. A molecular function describes the basic activity of a gene product that occurs at molecular level [21].

The ontologies are structured so that they can be queried at different levels, depending on the depth of knowledge about the considered entity. GO project offers the possibility to search, browse and visualize Gene Ontology data through the browser AmiGO, that is reachable from *http://amigo.geneontology.org/amigo*.

Currently, Gene Ontologies contain 41218 terms (39403 not obsolete). The set of terms includes 26840 (26048) biological processes, 3689 (3537) cellular component and 10657 (9750) molecular functions.

Figure 28 illustrates an example of gene product annotation.

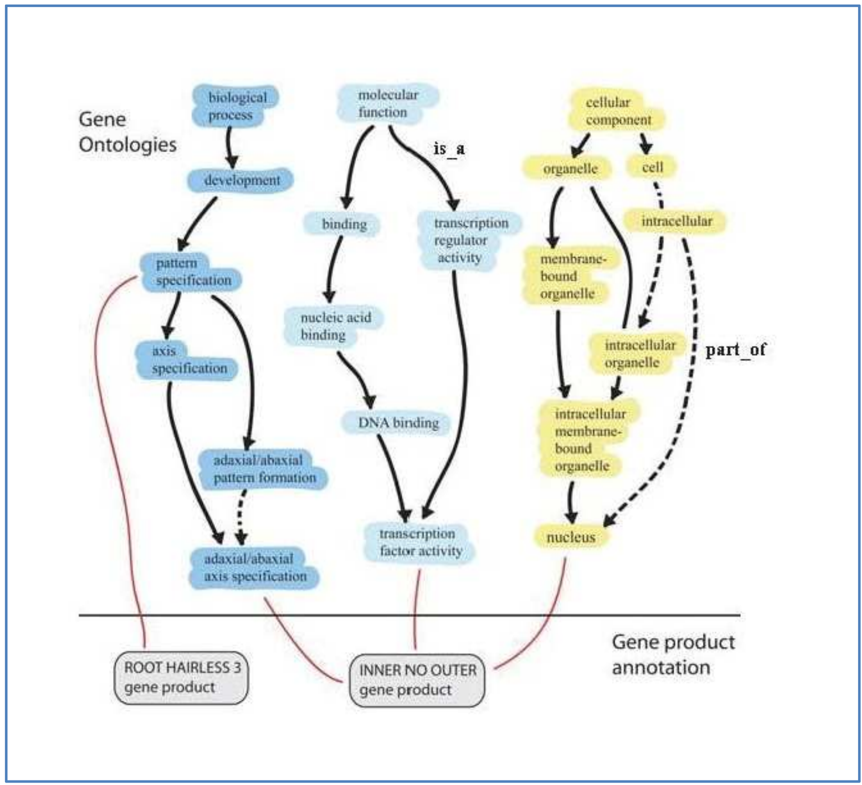


Figure 28 - Example of GO gene product annotation

GO databank provides all the ontological data into a single file that can be downloaded in different file format such as OBO, MySQL and SQL dump, RDF-XML, OBO-XML and OWL. In GPDW project, the choice of analyzing the OBO-XML file *go_daily-termdb.obo-xml.gz* requires the implementation of a new parser class, described in section 1, Chapter 7.

The ER schema designed for GO data source is illustrated in Figure 29.

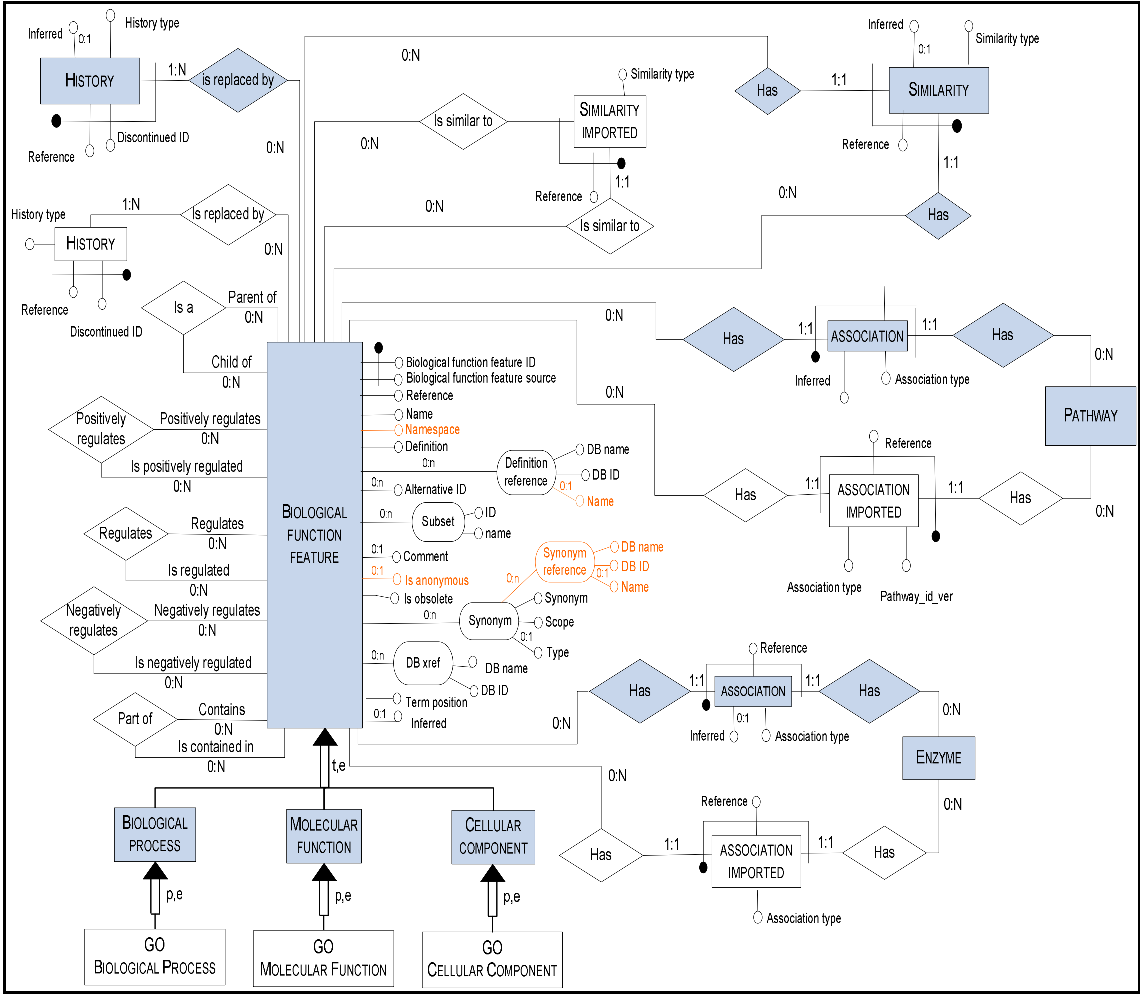


Figure 29 - GO ER schema

The complexity of the logical diagram and the high number of tables it contains create difficulties in representing the complete Logical schema. For this reason, the following figures will show the main parts of the designed schema separately.

Figure 30 introduces imported source tables for the sub-feature *cellular component*; for the other sub-features, *molecular function* and *biological process*, the same tables, with equal field names and number, are filled. In Figure 31 the association imported and integrated tables are shown. All the integrated tables of *biological function feature*, its history and similarity tables are displayed in Figure 32.

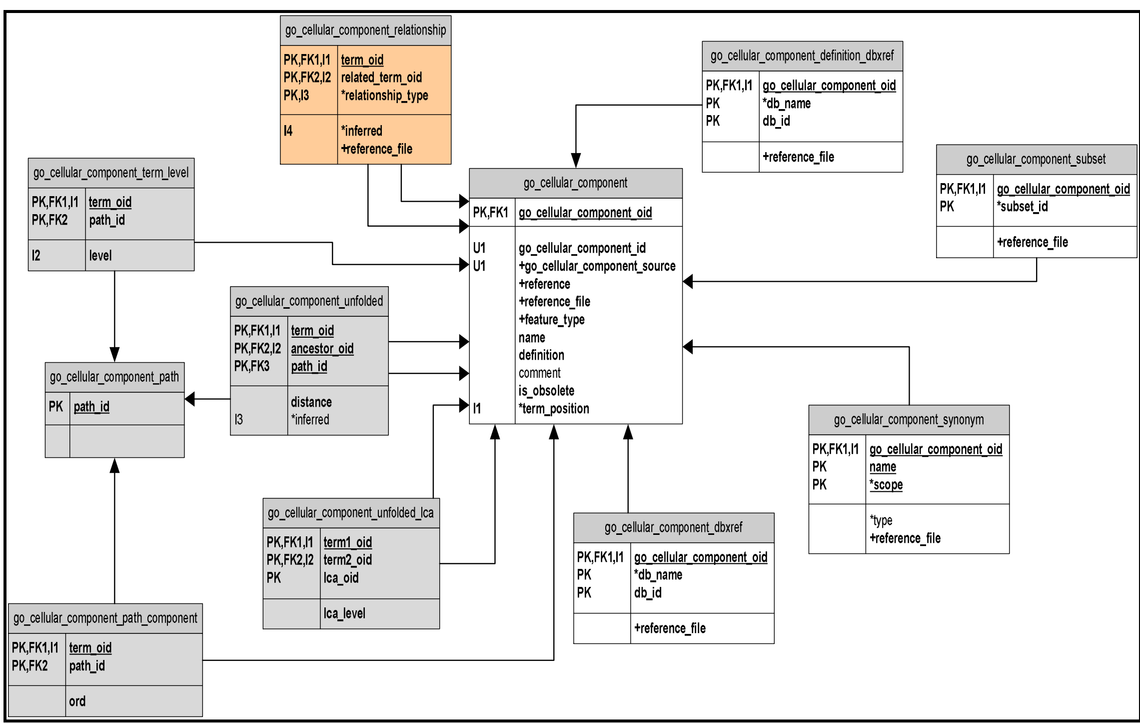


Figure 30 - GO Logical schema - cellular component source tables

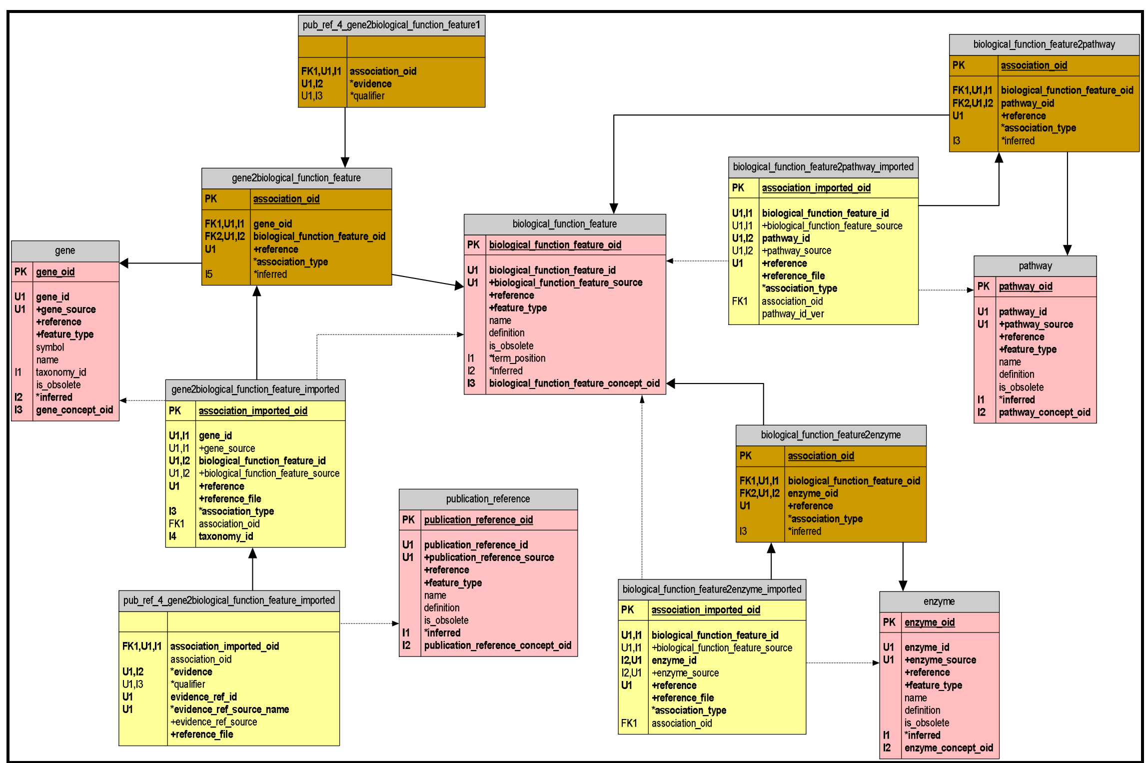


Figure 31 - GO Logical schema - association tables

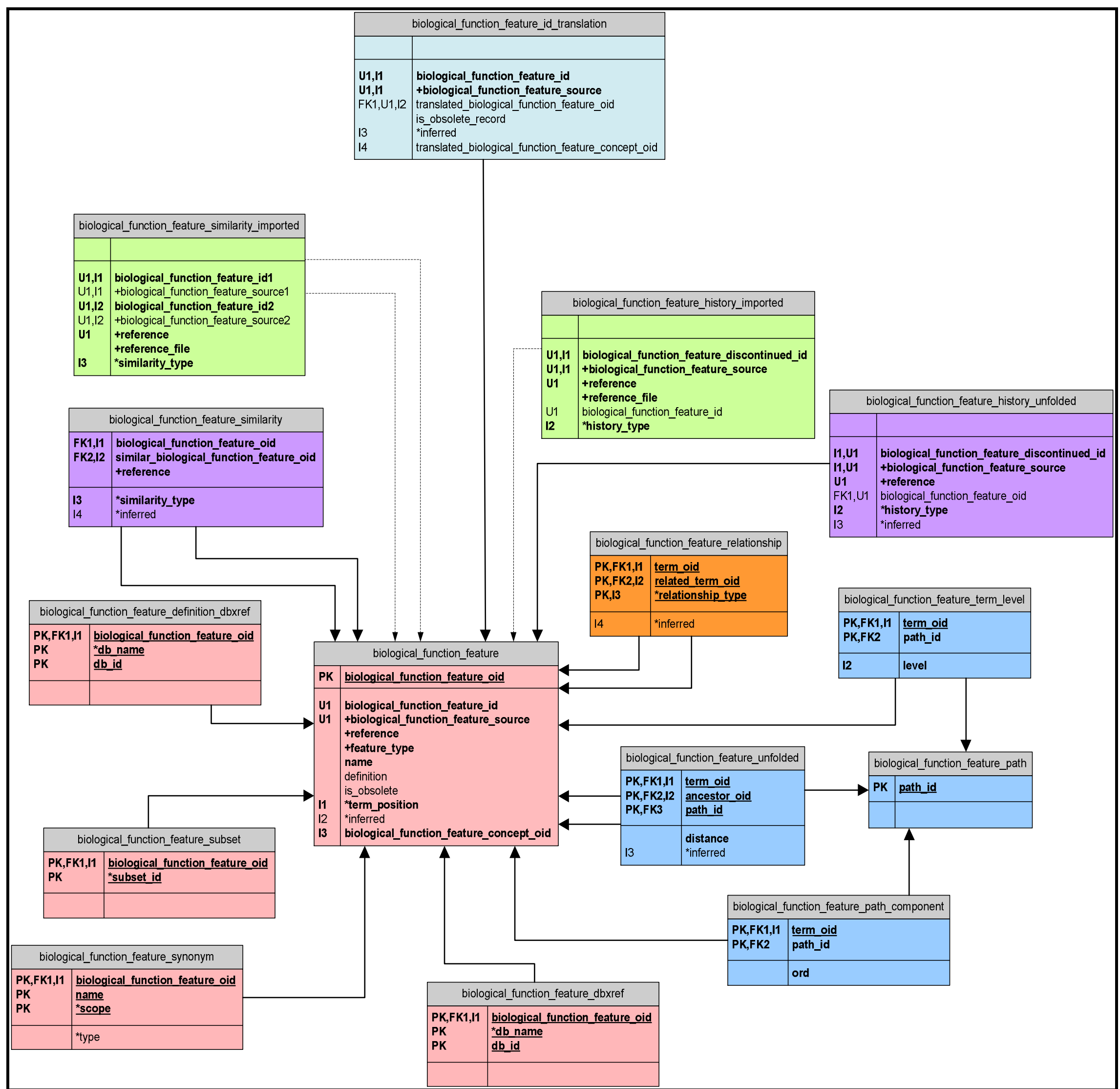


Figure 32 - GO Logical schema - integrated source tables

6.2 Gene Ontology Annotation - GOA

The Gene Ontology Annotation (GOA) database aims to provide high-quality electronic and manual annotations to the UniProt Knowledgebase (Swiss-Prot, TrEMBL and PIR-PSD) using the standardized vocabulary of the Gene Ontology [28]. It is the largest and most comprehensive open-source contributor of annotations to the GO Consortium annotation effort, providing annotated entries for nearly 60000 species.

GOA project distributes, in addition to a non-redundant set of annotations to the human proteome (GOA-Human) and monthly releases of its GO annotation for all species (GOA-SPTr), a series of GO mapping files and specific cross-references in other databases. The electronic assignment of GO terms to UniProt entries has been made possible by successfully converting part of the existing knowledge held within the flat files into GO terms.

For example, UniProt data may contain references to Enzyme Commission (EC or ExPASy Enzyme) numbers. Using an existing mapping of EC numbers to the GO molecular function ontology (ec2go) and a mapping of protein accession numbers to EC numbers, GOA can produce a UniProt to GO association.
In order to provide more reliable and specific annotation, GOA also makes use of manual curation using information extracted from scientific literature. Manual annotation is high-quality and reliable because it is validated by a team of skilled biologists.
Each assigned term is associated with a GO experimental evidence code and a PubMed ID, which allows users to track the literature source and type of experiment used to support the annotation.
There are various ways of accessing and searching GOA project information. In addition to several web-based browsers, GOA files and mappings can be downloaded from *ftp://ftp.ebi.ac.uk/pub/databases/GO/goa/*.

The list of GOA files considered in GPDW project contains:

- *ec2go*: mapping of EC numbers to GO terms;
- *interpro2go*: mapping of InterPro entries to GO terms;
- *pfam2go*: mapping of Pfam entries to GO terms;
- *gp2protein.geneid*: mapping of Entrez Gene ids to UniProtKB entries;
- *gene_association.goa_<species>* (where *<species>* can be uniprot, human, arabidopsis, chicken, cow, mouse, rat, zebrafish): it contains all GO annotations and protein information for a species subset of proteins in the UniProtKB.

gp2protein.geneid and *gene_association.goa_** are tabular files with header while *pfam2go, ec2go*, and *interpro2go* are tabular files with header lines and multiple field separator.
In file *pfam2go*, that is imported by class *PfamToGoLoader.java,* the fields in the file are separated by characters ' > ' and ' ; ' as shown in Figure 33.

```
!version date: 2014/06/07 14:46:03
!description: Mapping of GO terms to Pfam entries. This mapping is generated from data
supplied by InterPro for the InterPro2GO mapping.
!external resources: http://www.ebi.ac.uk/interpro, http://pfam.sanger.ac.uk/
!citationÂ : Hunter et al. (2009) Nucleic Acids Res. 37Â :D211-D215
!contact:interhelp@ebi.ac.uk
!
Pfam:PF00001 7tm_1 > GO:G-protein coupled receptor activity ; GO:0004930
Pfam:PF00001 7tm_1 > GO:G-protein coupled receptor signaling pathway ; GO:0007186
Pfam:PF00001 7tm_1 > GO:integral component of membrane ; GO:0016021
Pfam:PF00002 7tm_2 > GO:G-protein coupled receptor activity ; GO:0004930
Pfam:PF00002 7tm_2 > GO:G-protein coupled receptor signaling pathway ; GO:0007186
Pfam:PF00002 7tm_2 > GO:integral component of membrane ; GO:0016021
```

Figure 33 - Example of records in file pfam2go

Header lines, identified by the exclamation mark at the beginning, are not imported. The analysis of the first line after the header allows extracting the following fields:

- *Field1: Pfam id = Pfam:PF00001 7tm_1*
  Identifier assigned by Pfam data source.
- *Field2: GO name = GO:G-protein coupled receptor activity*
  Name of the GO term (field not imported).
- *Field3: GO id = GO:0004930*
  The identifier of the GO term.

The source identifiers of protein domain and biological function feature are matched to the regular expressions provided by the related data source definitions in *GPDW_definition.xml*. The table *biological_function_feature2protein_fam_dom_imported* is populated by parsing this file.

The latest version of GOA UniProt released on 10 June, 2014 and assembled using the publicly released data available in the source databases on 07 June, 2014 is briefly described in Table 3. The original Table is available from: *http://www.ebi.ac.uk/GOA/uniprot_release*.

GOA annotation source	Number of associations	Number of distinct proteins
Electronic GO annotation using InterPro to GO mapping	145931441	43005436
Electronic GO annotation using EC to GO mapping	6941248	6610288
Total Electronic GO annotation	304393501	45287137
Manual GO annotation by UniProt	256372	44478
Manual GO annotation by Reactome	72113	7663
Manual GO annotation by IntAct	53802	13506
Total Manual GO annotations	1526919	269947
Total GOA annotations	306020420	45372323
Total number of PubMed references	937423	

Table 3 - Associations in gene_association.goa_uniprot

The ER schema designed for GOA data source is illustrated in Figure 34. GOA Logical schema displayed in Figure 35 shows the annotation (association) between GO biological function feature and the following features:

- Protein from IPI, UniProt, RefSeq and Ensembl;
- Enzyme from ExPASy ENZYME;
- Protein families and domains from InterPro and Pfam.
- Annotation between gene (Entrez Gene) and protein (UniProt) are inserted into the table *gene2protein_imported*.

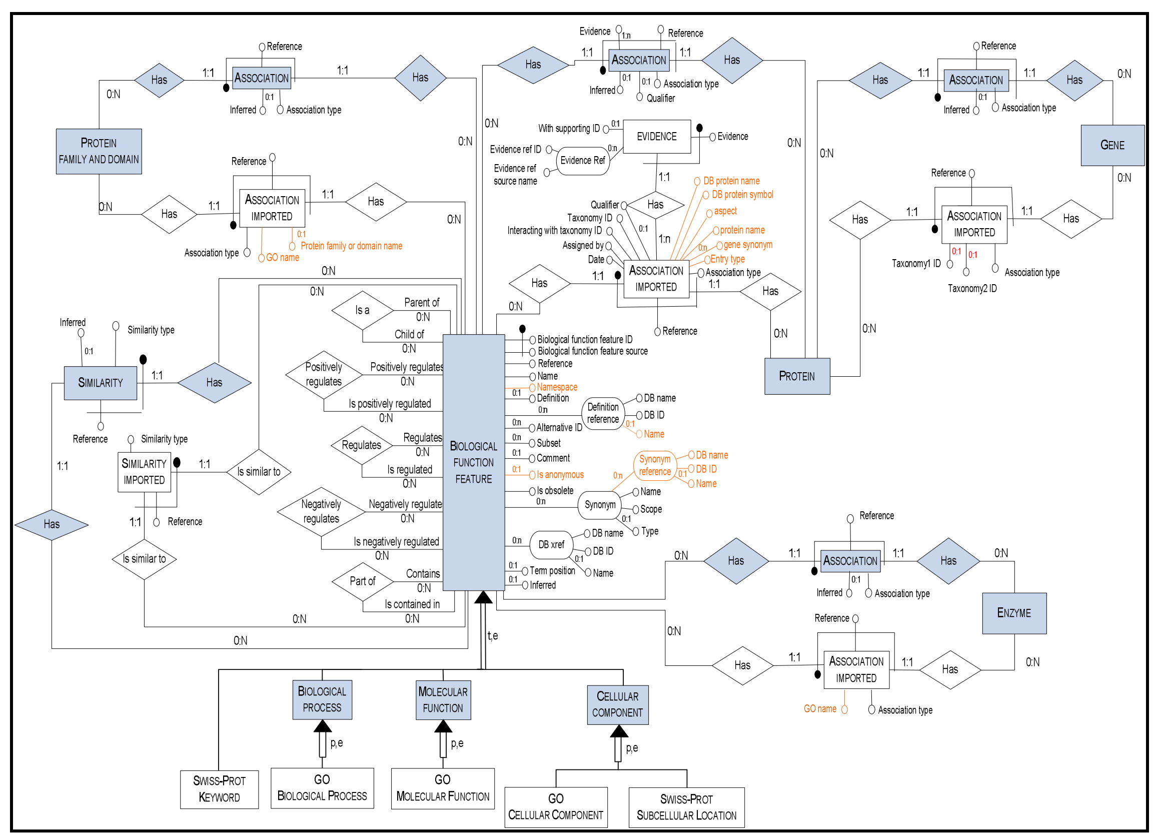


Figure 34 - GOA ER schema

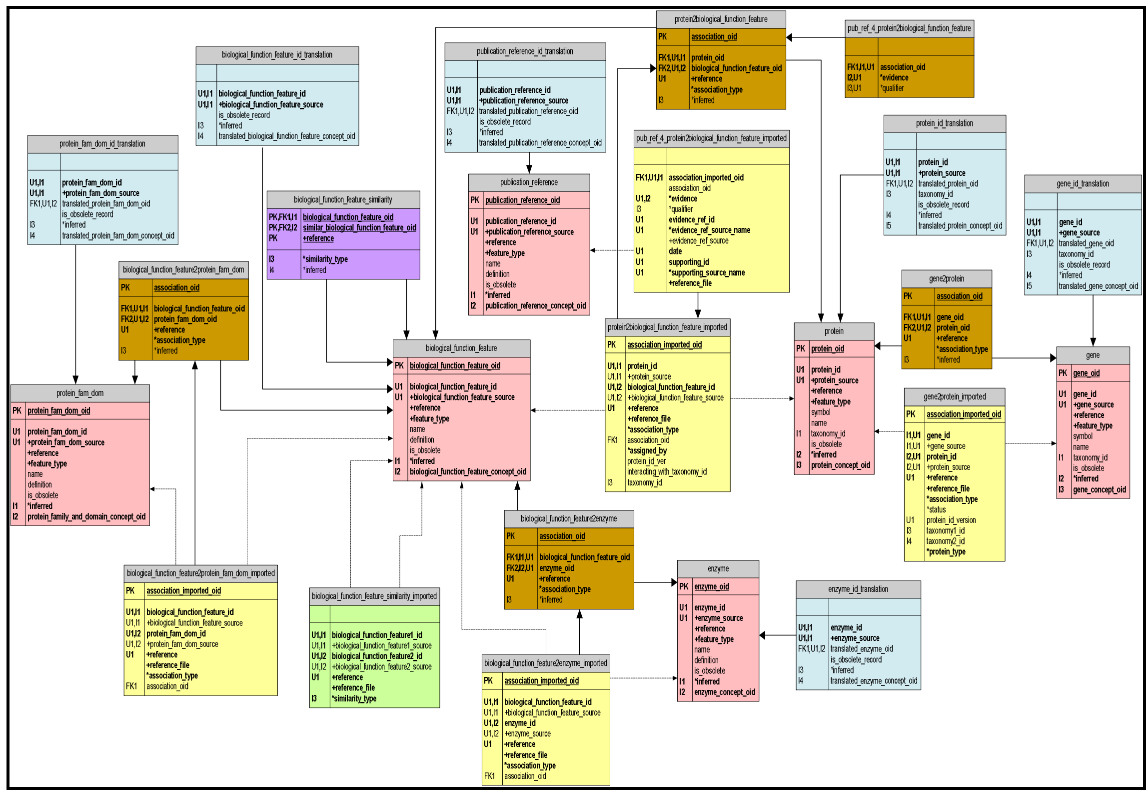


Figure 35 - GOA Logical schema

6.3 Entrez Gene

Entrez Gene is one of the several online databanks provided by NCBI (National Center for Biotechnology Information), founded in 1988 at US National Institute of Health (NIH).
Entrez Gene aims to provide tracked, unique identifiers for genes and to report information associated with those identifiers for unrestricted public use; it focuses on the genomes that have been completely sequenced or that are scheduled for intense sequence analysis [29]. The content of Entrez Gene stretches from gene products and map locations to phenotypes, sequences, variation details, homologs and protein domains.
Entrez Gene provides unique integer identifiers for genes and other loci for a subset of model organisms. These identifiers (GeneIDs) are specific for each species, i.e. an integer assigned to a protein in human is different from that in any other species. Identifiers are assigned to what is annotated as a gene as result of automated integration of data from NCBI's Reference Sequence project (RefSeq) and from many other databases from NCBI.
Currently, the database contains more than 7 million GeneID records distributed among more than 7300 taxonomies, most of which about viruses (~ 2400), archea/bacteria (~ 2300) and eukaryotes (~ 2300).
Entrez Gene provides a unified environment that allows to query the database and search genes by name, symbol, accessions, publications, chromosome numbers, GO terms, EC numbers. In Figure 36 a snapshot shows the results of search for *human muscular dystrophy*.

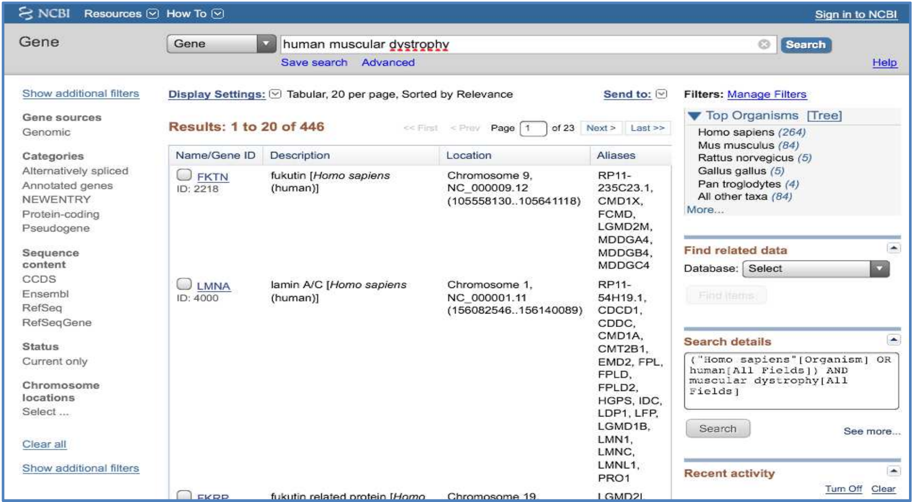


Figure 36 - Query results for human muscular dystrophy in Entrez Gene

Only the file *gene2go* was considered for the development of the Thesis. It is a tabular file with header lines and it reports the GO terms that have been associated with genes in Entrez Gene. This file, that is available from *ftp://ftp.ncbi.nih.gov/gene/DATA/gene2go.gz*, is automatically and daily recalculated by processing the gene_association files on the GO ftp website *http://www.geneontology.org/GO.current.annotations.shtml*.
The loader that manages its import is the class *GeneToGoLoader.java*. An example of records contained in the file is shown in Figure 37.

#Format: tax_id GeneID GO_ID Evidence Qualifier GO_term PubMed Category (tab is used as a separator, pound sign - start of a comment)							
3702	814629	GO:0003676	IEA	-	nucleic acid binding	-	Function
3702	814629	GO:0005575	ND	-	cellular_component	-	Component
3702	814629	GO:0008150	ND	-	biological_process	-	Process
3702	814629	GO:0008270	IEA	-	zinc ion binding	-	Function
3702	814630	GO:0003700	ISS	-	transcription factor activity	11118137	Function
3702	814630	GO:0045449	TAS	-	regulation of transcription	11118137	Process

Figure 37 - Example of records in file gene2go

The header line, identified by the hash at the beginning, is compared to the list of header parameters, contained in the variable *headerParams* in *GeneToGoLoader.java,* through the method *headerControl()* in order to check if errors in file format exist. By analyzing the last line in the figure, it is possible to identify eight fields:

- *Field1: tax_id = 3702* (Arabidopsis)
  Unique identifier provided by NCBI Taxonomy for the species.
- *Field2: GeneID = 814630*
  Unique identifier for a gene.
- *Field3: GO ID = GO:0045449*
  GO term identifier.
- *Field4: Evidence = TAS*
  Evidence code in the gene_association file.
- *Field5: Qualifier =* - (not present in this case)
  A qualifier for the association between the gene and the GO term.
- *Field6: GO term = regulation of transcription*
  Name of the term indicated by the *GO ID*. This field is not imported.
- *Field7: PubMed = 11118137*
  Pipe-delimited set of PubMed ids reported as evidence for the association.
- *Field8: Category = Process*
  GO category (Function, Process, or Component). This field is not imported.

Tables that are populated by parsing this file are *gene2biological_function_feature_imported* and *pub_ref_4_gene2biological_function_feature_imported.*
The Entity Relationship and Logical schemas shown in Figure 38 and Figure 39 respectively include only the association data considered by this file and information about source attributes, history and similarity.

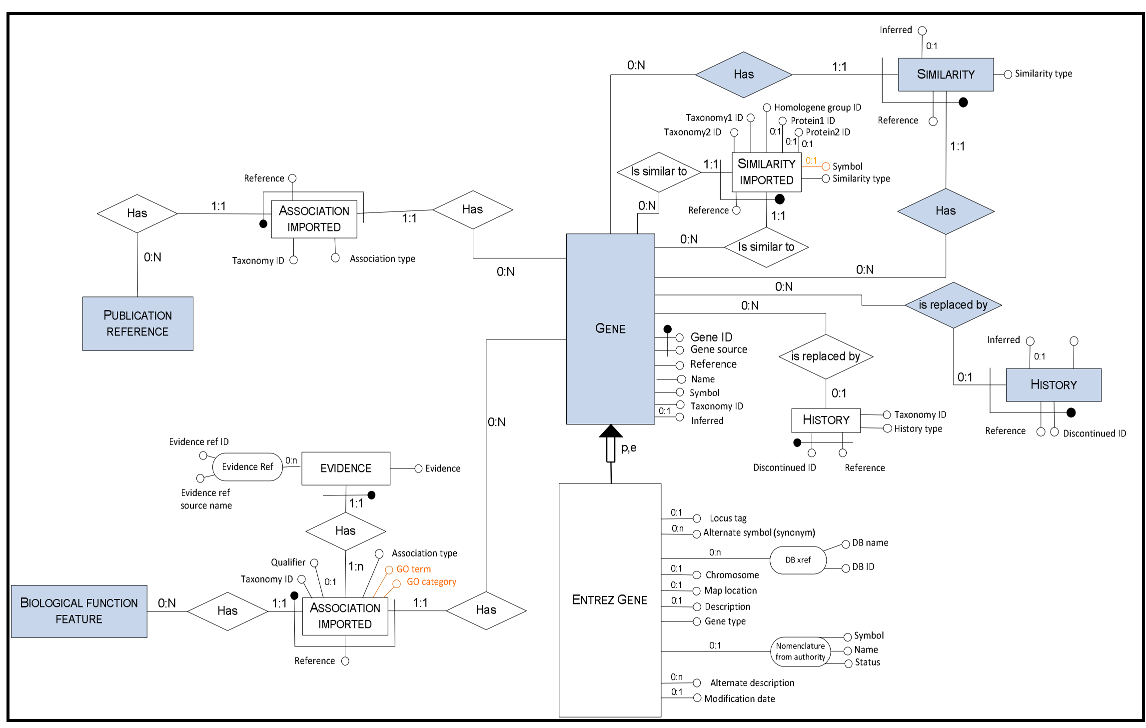


Figure 38 - Entrez Gene ER schema

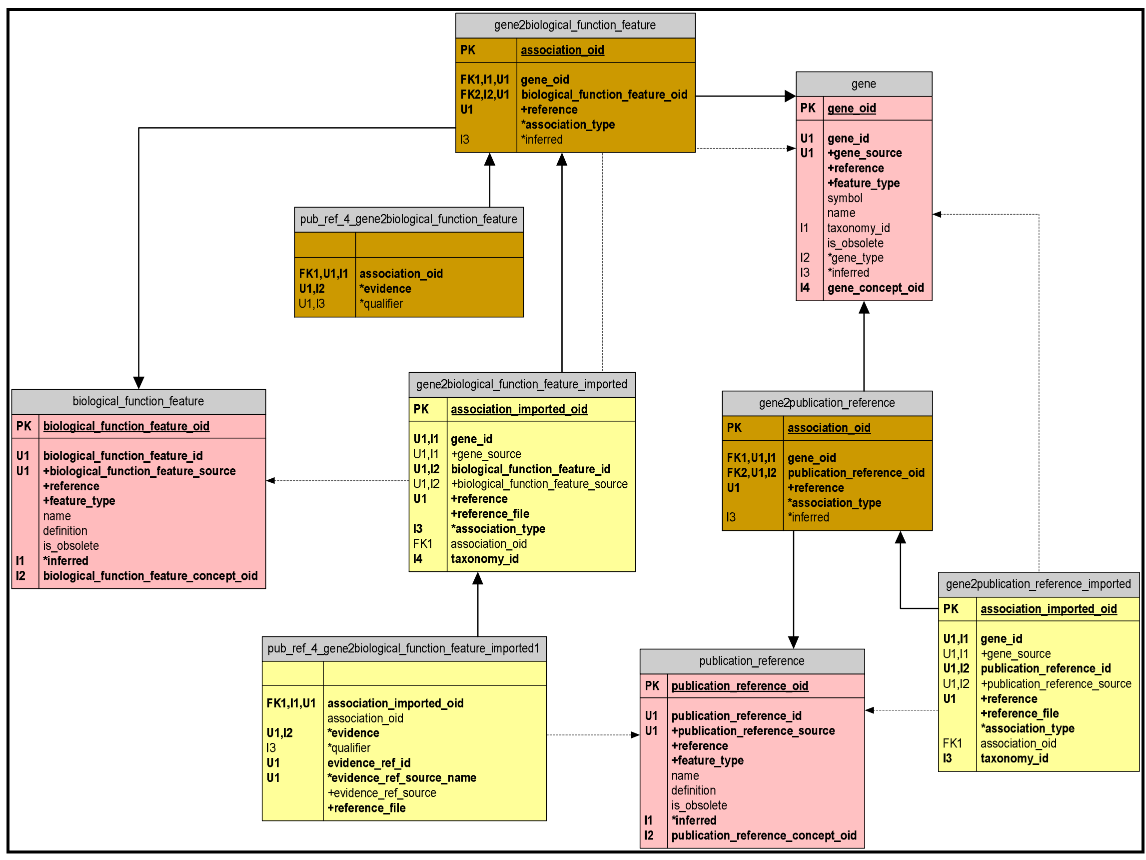


Figure 39 - Entrez Gene Logical schema

6.4 Expert Protein Analysis System ENZYME - ExPASy Enzyme

ExPASy (Expert Protein Analysis System) ENZYME is the Swiss Institute of Bioinformatics (SIB) repository of information about nomenclature of enzymes [30][31].
ExPASy was established in 1993 to offer to the scientific community a new instrument for the analysis of protein sequences and the prediction of their tertiary structure.
ExPASy server allows access to several genomic and proteomic data sources, with particular attention to their integration and cooperation. Indeed, ExPASy plays as main host of following databanks, mainly promoted by SIB in Geneva:

- SWISS-PROT
- SWISS-2DPAGE
- PROSITE
- ENZYME
- SWISS MODEL

ExPASy ENZYME describes each type of characterized enzyme for which an EC (Enzyme Commission) number has been provided, giving information about alternative names, catalytic activities, cofactors and pointers to human diseases associated to a deficiency of the enzyme. Source files of ExPASy ENZYME, available from *ftp://ftp.expasy.org/databases/enzyme/*, are:

- *enzclass.txt*, that contains the international hierarchical classification of enzymes, providing identifiers of enzymes classes, subclasses and sub-subclasses;
- *enzyme.dat,* that contains the complete database of enzymes ordered by EC number.

The loader that manages the import of tabular file *enzclass.txt* is the object *EnzymeClassLoader.java*, that extends the parser *TabularFileWithFixLengthHeaderParser.java*. The first records of the file are shown in Figure 40.

```
---------------------------------------------------------------------
ENZYME nomenclature database
Swiss Institute of Bioinformatics (SIB). Geneva, Switzerland
---------------------------------------------------------------------
Description:  Definition of enzymes classes, subclasses and sub-
Subclasses
Name:         ENZCLASS.TXT
Release of:   22-Jan-2014

---------------------------------------------------------------------

1. -. -.-  Oxidoreductases.
1. 1. -.-   Acting on the CH-OH group of donors.
1. 1. 1.-    With NAD(+)or NADP(+) as acceptor.
1. 1. 2.-    With a cytochrome as acceptor.
```

Figure 40 - Example of records in file enzclass.txt

The header is identified by a line of dashes and it contains information about file's version, authors and many additional data that are meaningless for the import operations. Header lines are compared to the list of parameters contained in the variable *headerParams* in *EnzymeClassLoader.java* and then they are discarded by overriding the method *onHeaderRead()*.
Hierarchical classification of enzymes is realized through a tree structure. For example, the subclass '*1. 1. -.-*' has parent class '*1. -. -.-*'. The use of string management's methods allows formatting enzyme ids by removing the last part of identifier that is made of repeated sequences of characters ' .- '.
The conceptual diagram for ExPASy ENZYME data source is displayed in Figure 41.

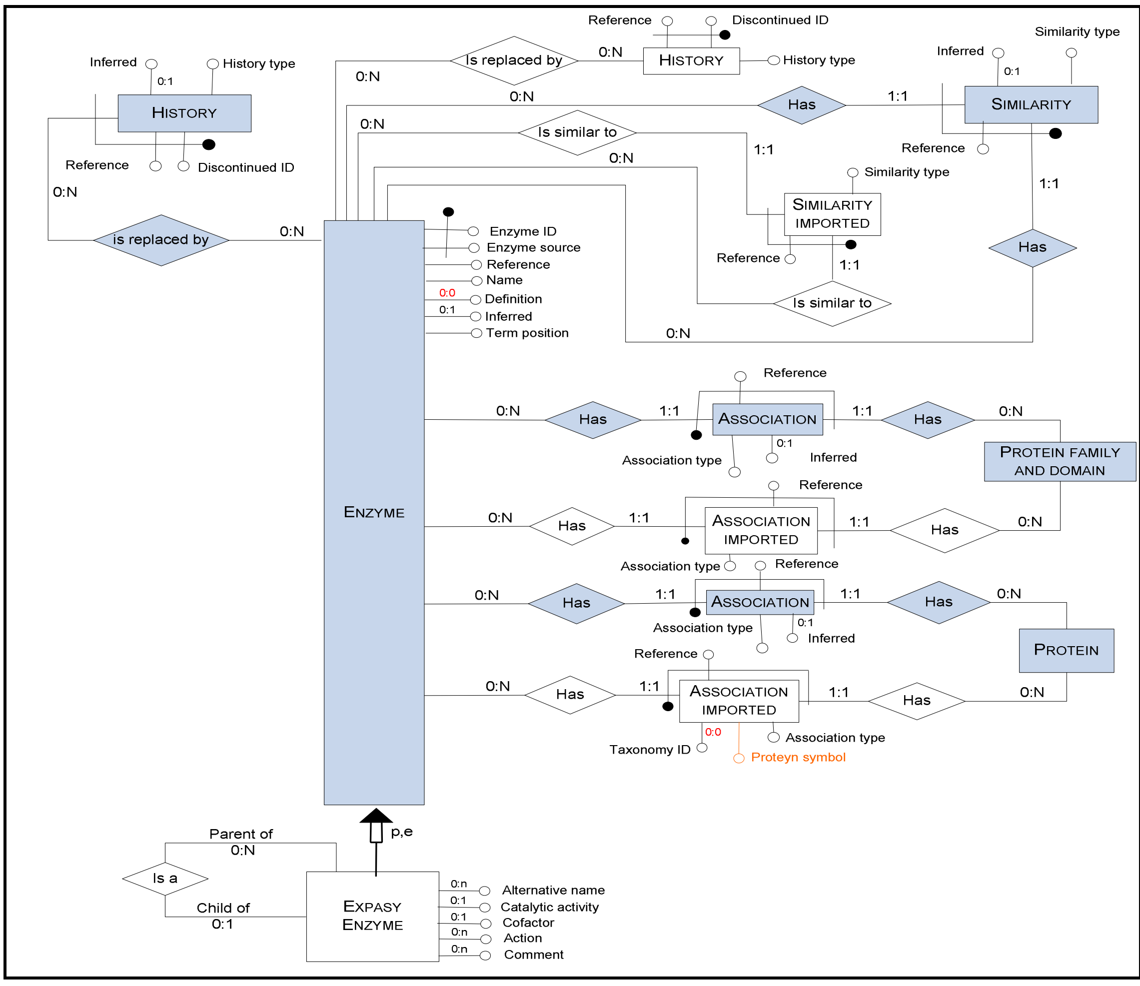


Figure 41 - ExPASy ENZYME ER schema

The parsing of *enzclass.txt* populates tables *expasy_enzyme* and *expasy_enzyme_relationship*. All the other tables that appear in Logical schemas in Figure 42 and Figure 43 are populated by the file *enzyme.txt*.

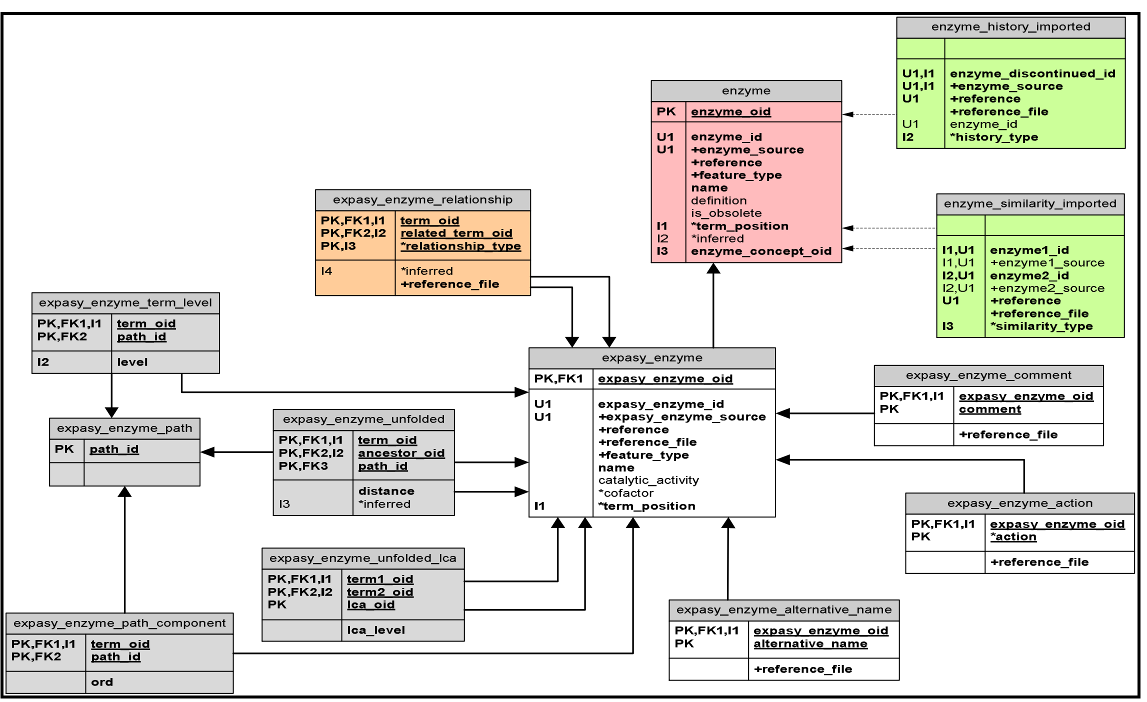


Figure 42 - ExPASy ENZYME Logical schema - imported tables

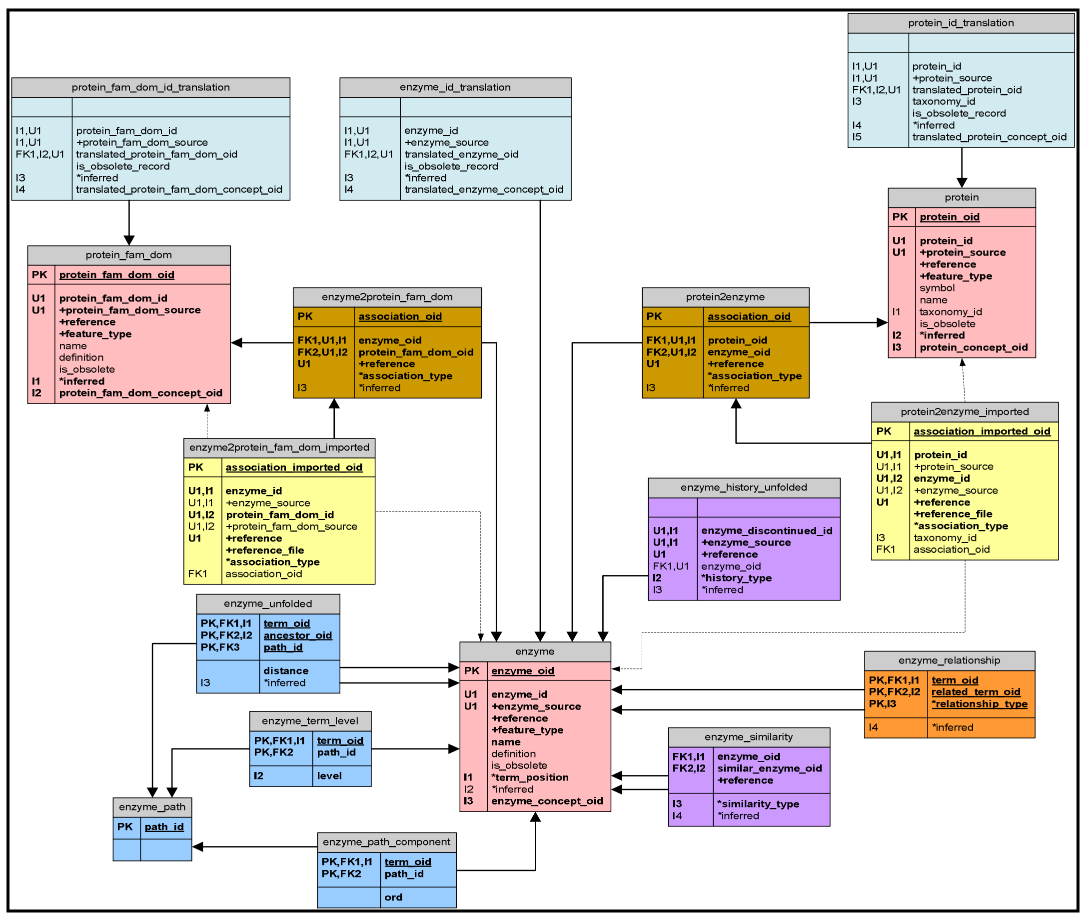


Figure 43 - ExPASy ENZYME Logical schema - integrated tables

6.5 Online Mendelian Inheritance in Man - OMIM

OMIM is a complete and daily updated catalog of human genes and genetic disorders, with links to scientific literature references, sequence records, maps and related databases, such as DNA, protein sequences and locus-specific mutation databases.
OMIM data provide information about all well-known Mendelian disorders, concentrating on relation between genotype and phenotype. OMIM data source started in 1960s thanks to the contribution of Dr. Victor A. McKusick.
Today OMIM is authored and edited at the McKusick-Nathans Institute of Genetic Medicine, Johns Hopkins University School of Medicine, under the direction of Dr. Ada Hamosh. The online version of OMIM, available on the WEB since 1987, is attended by the National Center Biotechnology Information (NCBI) [32][33]. Currently, OMIM data source provides more than 20000 entries, as shown in Table 4.

Number of entries in OMIM					
Prefix	Autosomal	X Linked	Y Linked	Mitochondrial	Totals
* Gene description	13859	675	48	35	14617
+ Gene and phenotype, combined	99	2	0	2	103
# Phenotype description, molecular basis known	3814	285	4	28	4131
% Phenotype description or locus, molecular basis unknown	1562	134	5	0	1701
Other	1738	115	2	0	1855
Totals	21072	1211	59	65	22407

Table 4 - Number of entries in OMIM

OMIM files considered in the GPDW data warehouse are:

- *omim.txt*: flat file containing all OMIM human genes and phenotypes description;
- *genemap.key:* flat file describing encoded fields of *genemap* file;
- *genemap*: list of OMIM genes ordered by cytogenetic location;
- *morbidmap*: list of alphabetical ordered genetic diseases in OMIM;
- *pubmed_cited*: it contains PubMed publication references about genes and disorders.

All these files are freely available from *ftp://ftp.ncbi.nih.gov/repository/OMIM/*. The complexity of both conceptual and logical diagrams makes their full representation quite difficult. The ER schema is split in two parts (Figure 44 and 45), while tables of the Logical schema are displayed in figures from 46 to 49.
The analysis of source files shows that there are two main ontological entities, *omim_gene* and *omim_disorder*, that are associated each other. There is also a small entity named *omim_clinical_synopsis* that is hierarchical self-related and associated to both main features.

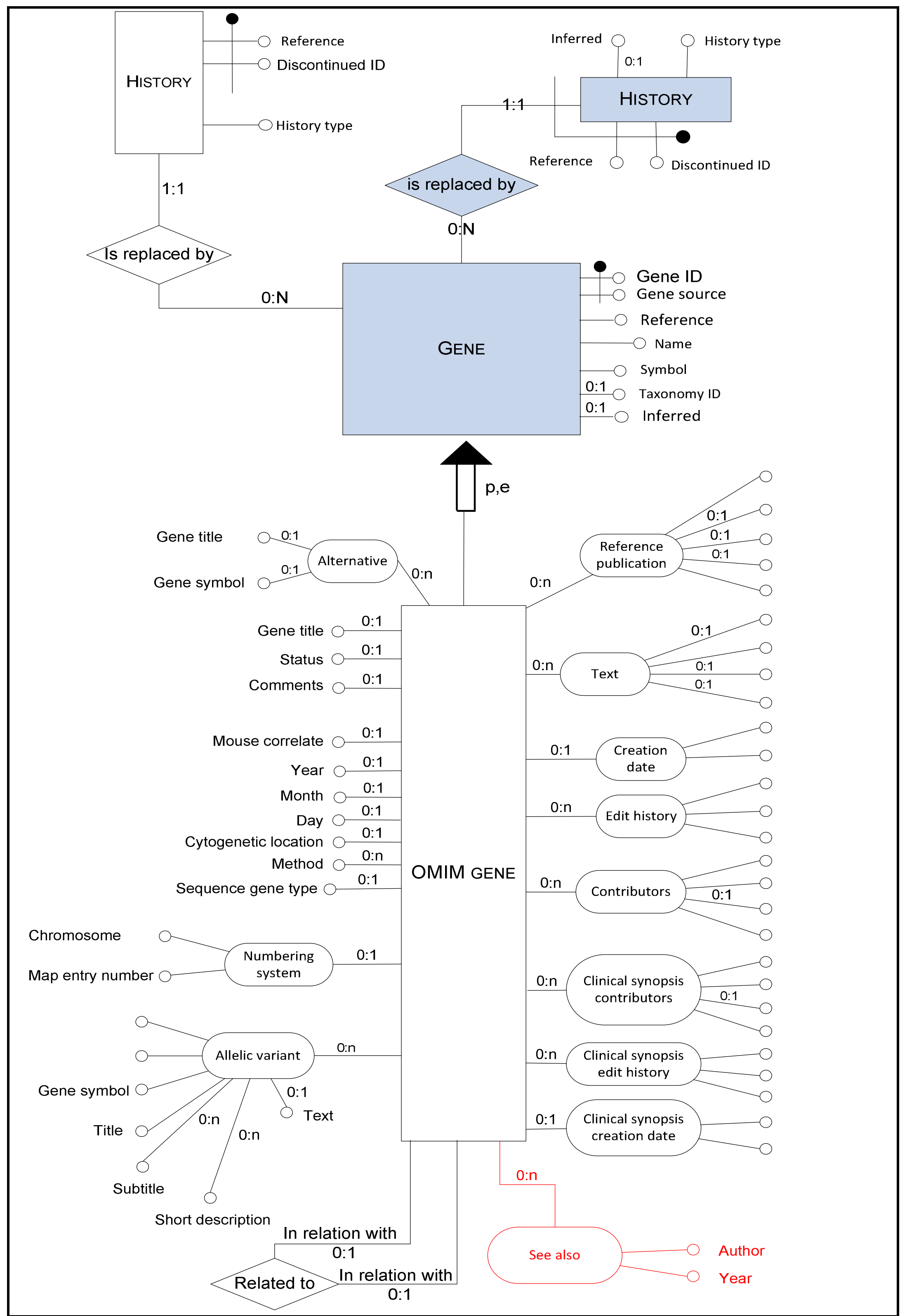


Figure 44 - OMIM ER schema - part I

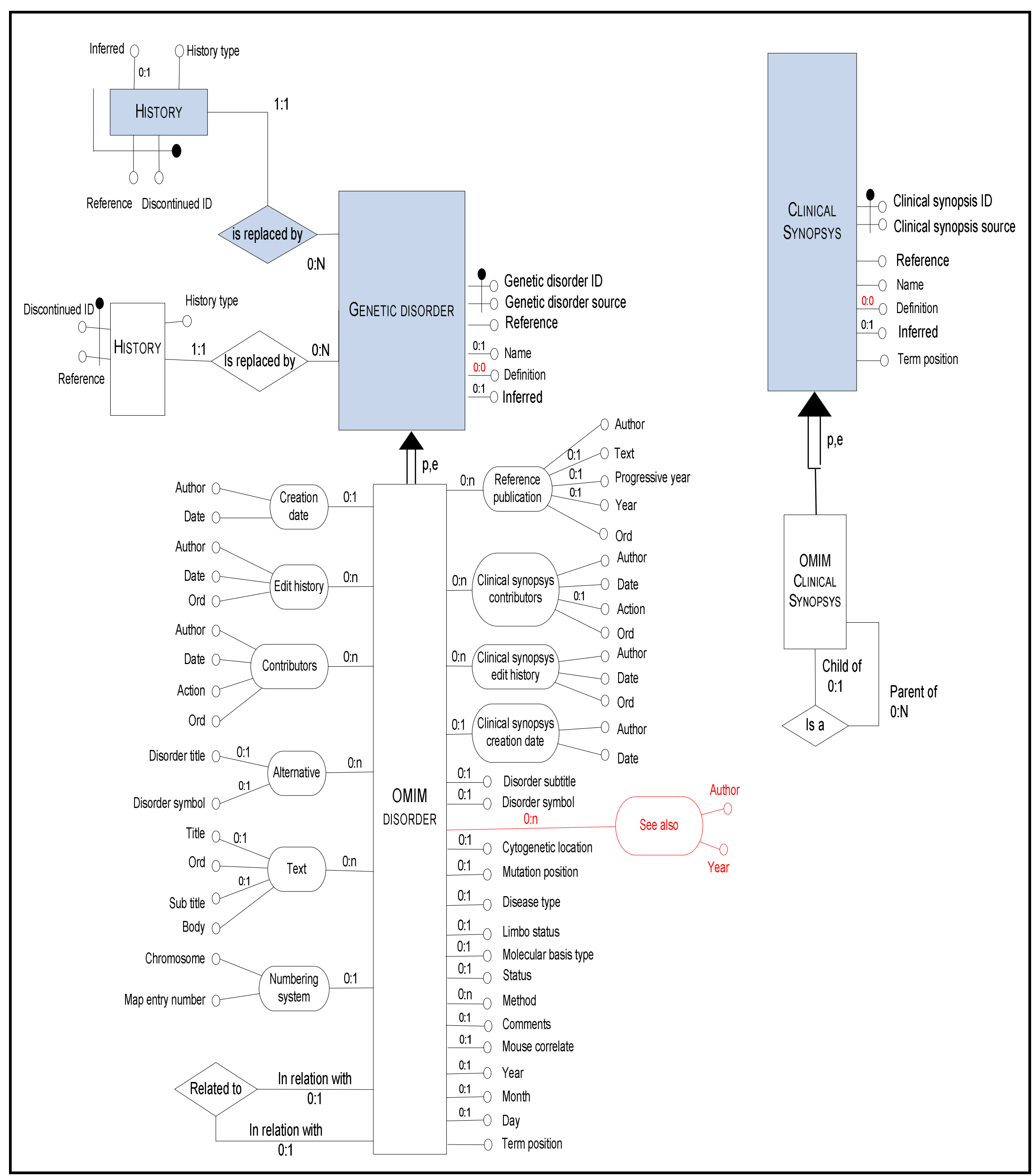


Figure 45 - OMIM ER schema - part II

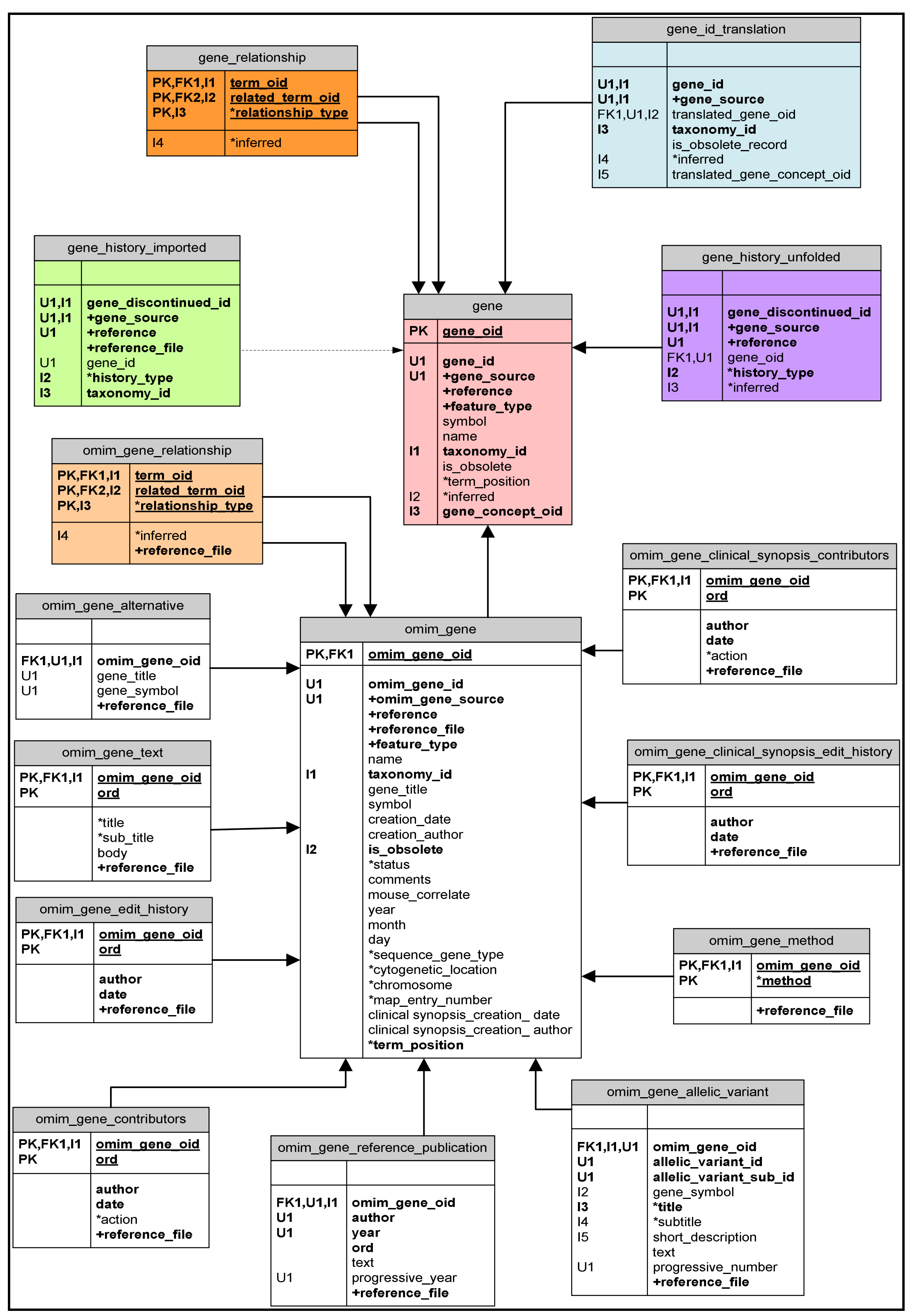


Figure 46 - OMIM Logical schema - gene

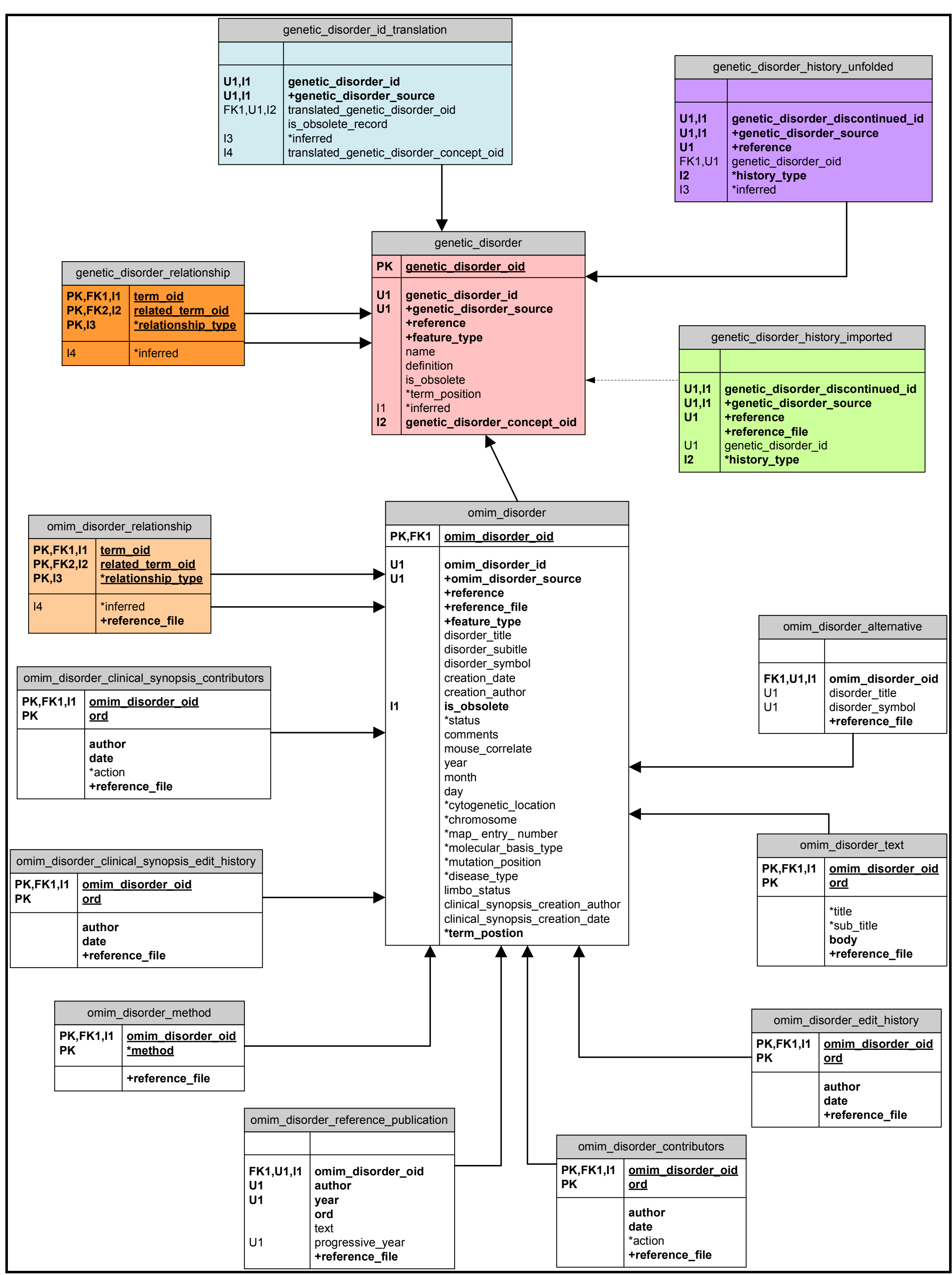


Figure 47 - OMIM Logical schema - genetic disorder

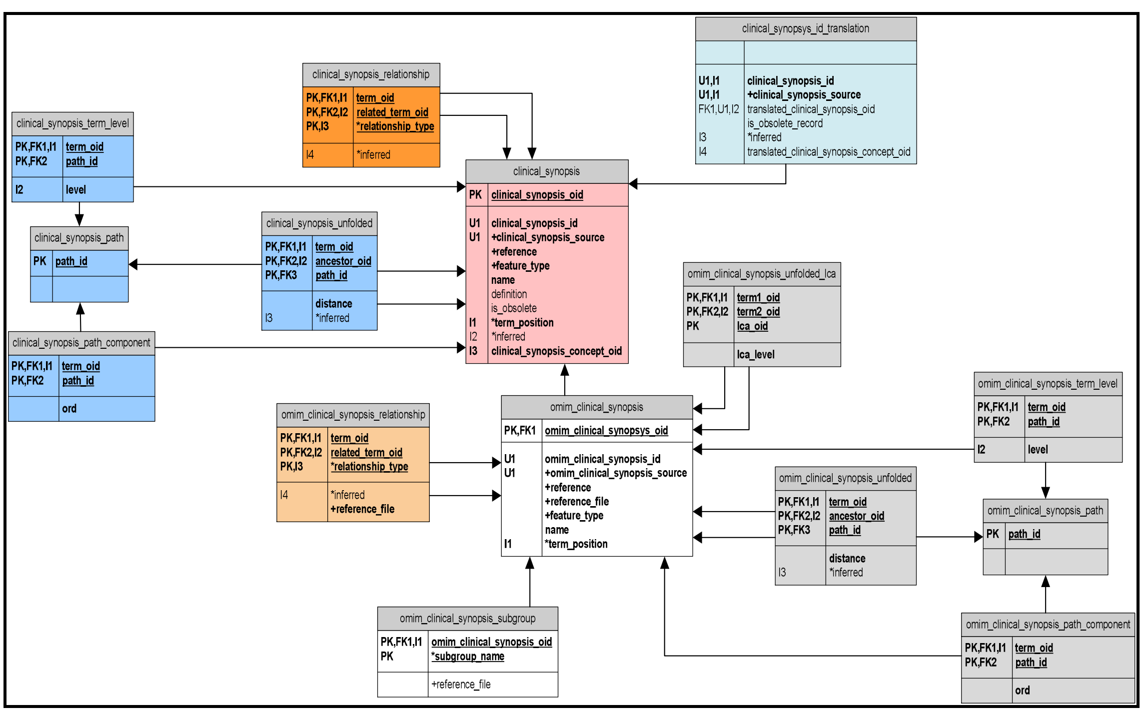


Figure 48 - OMIM Logical schema - clinical synopsis

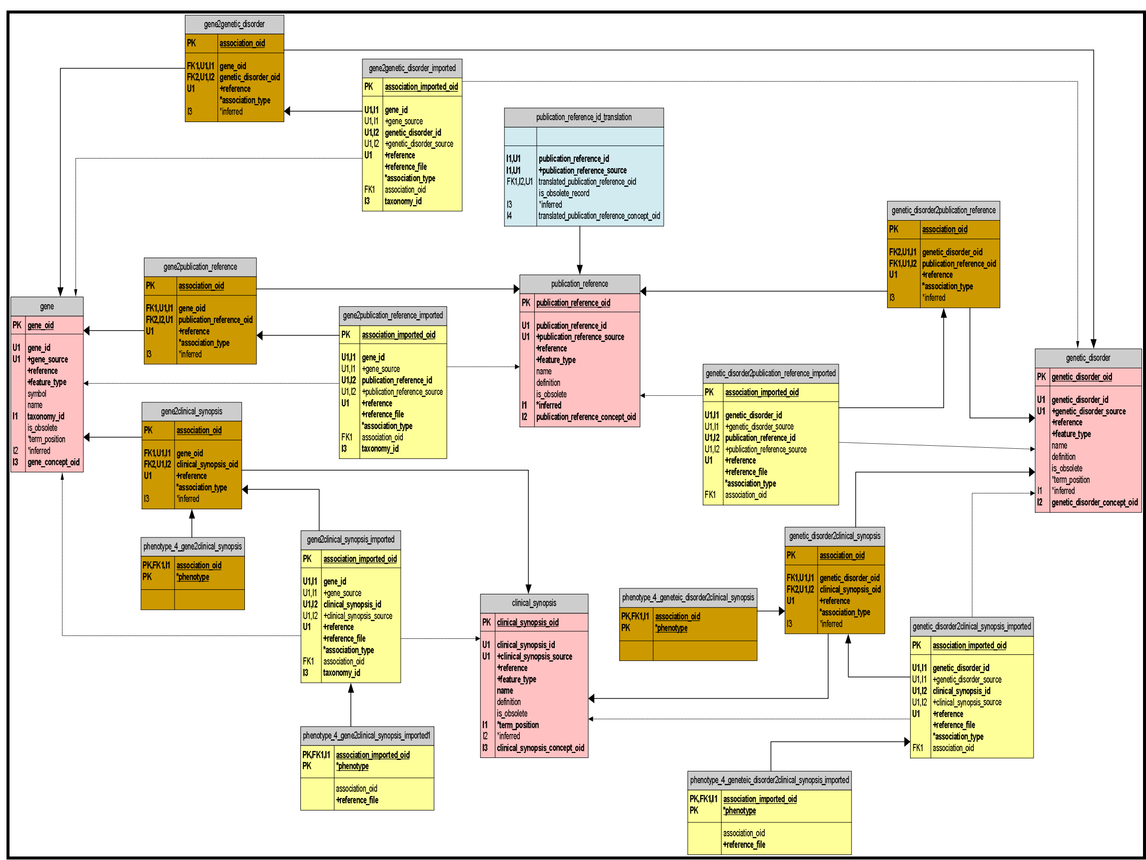


Figure 49 - OMIM Logical schema - association tables

7. Implementation of data import automatic procedures

This chapter is dedicated to the concrete implementation of the automatic procedures for importing the information provided by data sources previously described.
All the design choices and technical solutions adopted try to concretely implement computer science best practices in programming, software engineering and database management [34][35]. Specific use-cases are illustrated to prove the quality of introduced enhancements and the correctness of the operations performed by new modules described in the previous chapters.
The discussion focuses on the concrete implementation of the procedures for the alignment of source feature tables data import to the existing software framework.
The cases in which the format of data defined in source files do not exactly fit the generic procedures included in GPDW framework, so that the design and implementation of new components was required, are highlighted to prove the reliability of the resulting platform.

7.1 Parser implementation for go_daily-termdb.obo-xml data file

The import of *go_daily-termdb.obo-xml.gz* file required the implementation of a new parser to manage OBO-XML file's type. The parsers already present in the framework do not include any class that can process this file format; hence, before considering the import of GO file, it was created the generic abstract class *XmlFileParser.java*.
This object uses the java package *javax.xml.parsers* to parse the content of the input file as an XML document. An instance of *DocumentBuilder.java* class allows to parse the OBO-XML document and to return it as Document Object Model (DOM) from which a list of nodes is built. DOM is an object oriented model to represent structured documents. It is the W3C language independent standard for representing and interacting with objects in HTML and XML documents [36].
This model allows dynamically accessing content, structure and style of web document; the nodes of documents are organized in a tree structure, called the DOM tree, with topmost node named *Document object*. This type of implementation requires that the whole content of the document is analyzed and stored in the local memory, that is a computationally intensive process that can produce a considerable waste of memory in case of huge files. In case of GO, this choice is acceptable in relation to the dimension of the file that is approximately 50 MB.

The implemented Parser explores the list of nodes representing the DOM tree and "element" nodes are separately scanned by ad-hoc abstract methods according to their name:

- *source* - it contains metadata about the source file, such as its identifier, type and full path. These data are not relevant to the import process;
- *header* - it includes information about format version, release date, generation date and namespace of the current ontology. This element also contains the list of accepted subset and synonym type definitions;
- *term* - element that describes a GO term;
- *typedef* - element similar to *term*;
- *instance* - element similar to *term* but not currently used.

The manipulation of the data is left to the concrete implementation of the class, that is an instance of *GOLoader.java* in this case. The header tag is shown in Figure 50 and the semantic structure of element <term> is described in Figure 51.

```
<obo>
 <source>
  <source_id>/share/godev/goterm/... </source_id>
  <source_type>file</source_type>
  ...
 </source>
 <header>
  <format-version>1.2</format-version>
  <data-version>2014-06-14</data-version>
  <date>13:06:2014 07:57</date>
  <saved-by>dph</saved-by>
  <auto-generated-by> TermGenie 1.0</auto-generated-by>
  <subsetdef>
   <id>Cross_product_review</id>
   <name>Involved_in</name>
  </subsetdef>
  <subsetdef>
   <id>goslim_aspergillus</id>
   <name>Aspergillus GO slim</name>
  </subsetdef>
  ...
  <subsetdef>
   <id>virus_checked</id>
   <name>Viral overhaul terms</name>
  </subsetdef>
  <synonymtypedef>
   <id>systematic_synonym</id>
   <name>Systematic synonym</name>
   <scope>EXACT</scope>
  </synonymtypedef>
  <default-namespace>gene_ontology</default-namespace>
  ...
  <ontology>go</ontology>
  <property_value>propformat-version "1.2" xsd:string</property_value>
 </header>
```

Figure 50 - go_daily-termdb.obo-xml header

```
term =
   ## Terms are the fundamental units in an obo ontology
   ## Also known as classes
   ## Note: here 'term' refers to the primary term - synonyms are a
   ## a different representational entity
   element term {
     id
   & name
   & \namespace
   & def?
   & is_a*
   & alt_id*
   & subset*
   & comment?
   & is_anonymous?
   & is_obsolete?
   & consider*
   & replaced_by*
   & is_root?
   & xref_analog*
   & synonym*
   & relationship*
   & intersection_of*
   & union_of*
   & disjoint_from*
  }
```

Figure 51 - OBO ontology term

Below there is a detailed description of each tag of element <*term*>:

- **<id>** - The unique id of the current term; it is a seven digit identifier prefixed by "GO:" and it is usually called *term accession number*. The numerical portion of the ID has no inherent meaning or relation to the position of the term in the ontologies.
- **<name>** - The term name, e.g. mitochondrion, glucose transport, amino acid binding. Any term may have only one name defined. If multiple term names are defined, it is a parse error.
- **<namespace>** - Denotes which of the three sub-ontologies the term belongs to - cellular component, biological process or molecular function.
- **<def>** - A textual description of what the term represents, plus references to the source of the information. All new terms added to the ontology must have a definition; there remains a very small set of terms from the original ontology that lack definitions, but the vast majority of terms are defined.
- **<is_a>** - Describes a sub-classing relationship between one term and another. The value is the id of the term of which this term is a subclass. A term may have any number of is_a relationships.
- **<alt_id>** - Defines an alternate id for the current term. A term may have any number of alternate identifiers.
- **<subset>** - Indicates a term subset to which the term belongs. The value of this tag must be a subset name as defined in a *subsetdef* tag in the file header. If the value of this

tag is not mentioned in a *subsetdef* tag, a parse error will be generated. A term may belong to any number of subsets.

- **<comment>** - A comment providing extra information about the term and its usage. There must be zero or one instances of this tag per term description. More than one comment for a term generates a parse error.
- **<is_anonymous>** - Whether or not the current term has an anonymous id.
- **<is_obsolete>** - Boolean value that indicates whether or not the term is obsolete. Obsolete terms must have no relationships and no *defined is_a*, *inverse_of*, *union_of*, *disjoint_from* or *intersection_of* tags.

```
<term>
  <id>GO:0003732</id>
  <name>snRNA cap binding</name>
  <namespace>molecular_function</namespace>
  <def>
   <defstr>OBSOLETE. Interacting selectively with....</defstr>
   <dbxref>
    <acc>mah</acc>
    <dbname>GOC</dbname>
   </dbxref>
  </def>
  <comment>This term was made obsolete...</comment>
  <is_obsolete>1</is_obsolete>
  <consider>GO:0000339</consider>
 </term>
```

Figure 52 - Example of GO obsolete term

- **<consider>** - Gives a term which may be an appropriate substitute for an obsolete term, but needs to be looked at carefully by a human expert before the replacement is done. This tag may only be specified for obsolete terms. A single obsolete term may have many consider tags. This tag can be used in conjunction with *replaced_by*.
- **<replaced_by>** - Gives a term which replaces an obsolete term. The value is the id of the replacement term. The value of this tag can safely be used to automatically reassign instances whose *instance_of* property points to an obsolete term. The *replaced_by* tag may only be specified for obsolete terms. A single obsolete term may have more than one *replaced_by* tag. This tag can be used in conjunction with the *consider tag.*
- **<is_root>** - Boolean value that may be set to true if the term is a root in the current ontology.
- **<xref_analog>** - A database cross-reference, or dbxref, that describes an analogous term in another vocabulary. A term may have any number of dbxrefs. For instance, the molecular function term *retinal isomerase activity* is cross-referenced with the Enzyme Commission (ExPASy ENZYME) entry *EC:5.2.1.3*;

- **<synonym>** - A synonym for the current term, together with cross-references to describe the origins of the synonym. This tag may also indicate a synonym category or scope information. The synonym scope may be one of four values: EXACT, BROAD, NARROW, RELATED. The synonym type must be the id of a synonym type defined by a *synonymtypedef* line in the header. If the synonym type has a default scope, that scope is used regardless of any scope declaration given by a synonym tag. A term may have any number of synonyms.

```
<term>
  <id>GO:0008152</id>
  ...
   <dbxref>
    <acc>0198547684</acc>
    <dbname>ISBN</dbname>
   </dbxref>
  </def>
  <comment>Note that...</comment>
  <subset>goslim_pir</subset>
  ...
  <synonym scope="narrow">
   <synonym_text>metabolic process resulting in cell growth</synonym_text>
  </synonym>
  <synonym scope="exact">
   <synonym_text>metabolism</synonym_text>
  </synonym>
  ...
  <xref_analog>
   <acc>Metabolism</acc>
   <dbname>Wikipedia</dbname>
  </xref_analog>
  <is_a>GO:0008150</is_a>
 </term>
```

Figure 53 - Example of GO synonym term

- **<relationship>** - Describes a typed relationship between the current term and another term. The value of this tag should be the relationship type id, and then the id of the target term. If the relationship type name is undefined, a parse error will be generated. If the id of the target term cannot be resolved by the end of parsing the current batch of files, this tag describes a "dangling reference". If a relationship is specified for a term with true value for is_obsolete , a parse error will be generated.

- **<intersection_of>** - Indicates that the term is equivalent to the intersection of several other terms. The value is either a term id, or a relationship type id, a space, and a term id. A collection of *intersection_of* tags appearing in a term is also known as a cross-product definition

- **<union_of>** - Indicates that the term represents the union of several other terms. The value is the id of one of the other terms of which this term is a union. If any *union_of* tags are specified for a term, at least 2 *union_of* tags need to be present or it

is a parse error. The full union for the term is the set of all ids specified by all *union_of* tags for that term.

- **<disjoint_from>** - Indicates that a term is disjoint from another, meaning that the two terms have no instances or subclasses in common. The value is the id of the term from which the current term is disjoint. This tag may not be applied to relationship types.

```
<term>
  <id>GO:0098538</id>
  <name>lumenal side of transport vesicle membrane</name>
  <namespace>cellular_component</namespace>
  <def>
    <defstr>The side (leaflet) of the transport vesicle membrane that faces the lumen.</defstr>
    <dbxref>
      <acc>ab</acc>
      <dbname>GOC</dbname>
    </dbxref>
  </def>
  <synonym scope="exact">
    <synonym_text>internal side of transport vesicle membrane</synonym_text>
  </synonym>
  <is_a>GO:0044433</is_a>
  <is_a>GO:0098576</is_a>
  <relationship>
    <type>part_of</type>
    <to>GO:0030658</to>
  </relationship>
 </term>
```

Figure 54 - Example of GO term's relationship

The three different gene products' categories (ontologies) provided by GO are included, in GPDW framework, as shown in the Logical diagrams supplied in Chapter 6, into the same major biomedical feature called *biological function feature*. Hence, in the importing layer, source tables are populated for each category, or sub-feature, as suggested by GO ontologies; in the aggregation step, however, all these information are integrated into database tables directly referring the main entity *biological function feature*.

7.2 Import of source tables

The following sections contain two examples of concrete implementation of the importing procedures described in this Thesis for the automatic import of source feature tables.
The population of imported enzyme tables of ExPASy ENZYME and imported association tables of GOA have been selected as valid examples of the correct integration of new components in the framework architecture.

7.2.1 ExPASy ENZYME enzyme.dat data file

The file *enzyme.dat* contains the complete ENZYME database. The object *EnzymeDataLoader.java* is the loader that instantiates the parser *FlatFileWithHeaderParser.java* to enable parsing operations for the import of data from this file. The code displayed in Figure 55 describes a typical record of *enzyme.dat*.

```
ID    1.1.1.2
DE    Alcohol dehydrogenase (NADP(+)).
AN    Aldehyde reductase (NADPH).
CA    An alcohol + NADP(+) = an aldehyde + NADPH.
CF    Zinc.
CC    -!- Some members of this group oxidize only primary alcohols;
CC    others act also on secondary alcohols.
CC    -!- May be identical with EC 1.1.1.19, EC 1.1.1.33 and EC 1.1.1.55.
CC    -!- Re-specific with respect to NADPH.
PR    PROSITE; PDOC00061;
DR    Q6AZW2, A1A1A_DANRE; Q568L5, A1A1B_DANRE; Q24857, ADH3_ENTHI ;
....
//
```

Figure 55 - Example of records in file enzyme.dat

Each line of the data record is identified by a label of two characters. The string '//' is the termination line and it indicates the end of the record. All the possible labels are listed in Table 5, where cardinality values are also specified.

Field label	Field name	Label cardinality per record	Values cardinality per record
ID	Identification	1:1	1:1
DE	Description	1:n	1:1
AN	Alternate Name(s)	0:n	0:1
CA	Catalytic Activity	1:n	0:1
CF	Cofactor(s)	0:n	0:1
CC	Comments	0:n	0:n
PR	PROSITE cross-references	0:n	0:n
DR	Database cross-references	0:n	0:n

Table 5 - Field labels in file enzyme.dat

Below, the record's fields are briefly described:

- *Field1: Identification*
  EC number is always the first line of an entry.

- *Field2: Description*
  It contains the NC-IUB recommended name for an enzyme. Obsolete EC numbers are indicated by the following DE line syntaxes:
  - *DE* *Deleted entry.* for removed enzymes;
  - *DE* *Transferred entry: x.x.x.x.* where *x.x.x.x* is the new valid EC number.

- *Field3: Alternate Name(s)*
  Alternative names, different than the recommended one, that are used in the literature to describe an enzyme.

- *Field4: Catalytic Activity*
  CA lines are used to indicate the reactions catalyzed by an enzyme. The majority of reactions are described through standard chemical reaction format, while in some cases free text is used.

- *Field6: Cofactor(s)*
  It indicates which cofactors are required by an enzyme. The format of CF lines is *Cofactor_1; Cofactor_2 or Cofactor_3[; Cofactor_N...].*

- *Field7: Comments*
  Free text comments that may be used to convey information on enzyme similarity, history and actions performed by enzymes, such as catalysis, hydrolysis, reduction and oxidation.

- *Field8: PROSITE cross-references*
  PR lines are used as pointers to the PROSITE document, specified by its accession number, that mentions the enzyme being described. The document entry must be validated by matching the regular expressions defined for PROSITE data source.

- *Field9: Database cross-references*
  DR lines are used as pointers to the UniProtKB/Swiss-Prot entries that correspond to the enzyme being described. The entry must be validated by matching the regular expressions defined for UniProt data source.

This meticulous analysis of the file allows the creation of objects of type String, List, Arrays or HashSet that will act as temporary containers for the data to insert into the specific Entry structures, *mainSrcTblEntries* and *addSrcTblEntries*, that are configured and managed by GenericLoader. In this way, the insertion of records in database tables become an automatic standard operation as shown in figures 56 and 57.

```java
/**
* This method creates entries for the expasy_enzyme source table.
*/
private void insertExpasyEnzyme(){

   if (!expasy.contains(sourceID)) {
        expasy.add(sourceID);

        mainSrcTblParams.get(EXPASYENZYME).get(NAME).setValue(enzymeName);
        if(catalyticActivity.matches(""))
           mainSrcTblParams.get(EXPASYENZYME).get(CATAL).setValue(null);
        else
           mainSrcTblParams.get(EXPASYENZYME).get(CATAL).setValue(catalyticActivity);
        if(cofactor.matches(""))
           mainSrcTblParams.get(EXPASYENZYME).get(COFACT).setValue(null);
        else
           mainSrcTblParams.get(EXPASYENZYME).get(COFACT).setValue(cofactor);

        entry = new MainSourceTableEntry(sourceID, reference, featureTypeEnzyme,
                        mainSrcTblParams.get(EXPASYENZYME),
                        mainSrcTblImporters.get(EXPASYENZYME).getAddSrcTblEntries());
        insertComment();
        insertAlternativeName();
        insertAction();
        mainSrcTblEntries.get(EXPASYENZYME).add(entry);
        enzymeCnt++;
    }
    else
        logger.warn("Duplicated entry: expasy_id: " + sourceID + " already exists.");
}
```

Figure 56 - Entry creation for expasy_enzyme table

```java
/**
* This method creates entries for the expasy_enzyme_action table
*/
private void insertAction(){

   if(action.size() > 0){
        Iterator<String> it = action.iterator();
        for (int i = 0; i < action.size(); i++) {

            mainSrcTblImporters.get(EXPASYENZYME).getAddSrcTblParams().
                                   get(EXPENZYACT).get(ACT).
                                   setValue(it.next().toString());
            AdditionalSourceTableEntry ent = new
            AdditionalSourceTableEntry(mainSrcTblImporters.get(EXPASYENZYME).
                                        getAddSrcTblParams().get(EXPENZYACT));

            entry.getAddSrcTblEntries().get(EXPENZYACT).add(ent);
            actionCnt++;
        }
   }
}
```

Figure 57 - Entry creation for expasy_enzyme_action table

7.2.2 GOA gene_association.goa_<species> data files

Loader class *GOAAssociationLoader.java*, that extends the class *TabularFileWithHeaderParser*, is used to import association data provided in the following files:

- *gene_association.goa_uniprot;*
- *gene_association.goa_human;*
- *gene_association.goa_arabidopsis;*
- *gene_association.goa_chicken;*
- *gene_association.goa_cow;*
- *gene_association.goa_mouse;*
- *gene_association.goa_rat;*
- *gene_association.goa_zebrafish.*

The file *gene_association.goa_uniprot* contains both electronic - computed by using mapping files - and manual - provided by several data sources - GO annotations. It includes unfiltered associations between gene products, GO terms and associated annotation information for all the species considered in UniProt Knowledgebase.
The remaining files provide species-specific annotation and they consider only the subset represented by individual species; for each file, the annotation set is filtered in order to reduce redundancy. All the files share the same tabular structure with a single delimiter represented by the tab character; the information is distributed into the following fields:

- *Field1: DB*
  Database from which annotated entity has been taken. This field is not imported.

- *Field2: DB_Object_ID*
  A unique identifier in the database for the item being annotated. Example: *O00165*.

- *Field3: DB_Object_Symbol*
  A unique and valid symbol (gene name) that corresponds to the *DB_Object_ID*. This field is not imported.

- *Field4: Qualifier*
  This column is used for flags that modify the interpretation of an annotation. The values that may be present in this field are:
  - NOT;
  - (NOT|) colocalizes_with;
  - (NOT|) contributes_to;

- *Field5: GO ID*
  The GO identifier for the term attributed to the *DB_Object_ID*.

- *Field6: DB:Reference*
  A single reference to support an annotation. Where an annotation cannot reference a paper, this field contains a *GO_REF* identifier [37]. Examples: *GO_REF:0000020, PMID:9058808*. This field is mapped into two columns, *reference_ref_id* and *evidence_ref_source_name* by separating the id from the source name;

- *Field7: Evidence Code*
  This column is used for one of the evidence codes supplied by the GO Consortium. Examples: *IDA*, *IEA*.

- *Field8: With (or) From*
  Additional identifier(s) to support annotations using certain evidence codes (including IEA, IPI, IGI, IMP, IC and ISS evidences). Examples: *InterPro:IPROO1878*, *EC:3.1.22.1*. This field is mapped into two table columns, *supporting_id* and *supporting_source_name* by separating the identifier from the name of the source;

- *Field9: Aspect*
  One of the three GO ontologies: P (biological process), F (molecular function) or C (cellular component). This field is not imported.

- *Field10: DB_Object_Name*
  The full UniProt protein name will be present here, if available from UniProtKB. If a name cannot be added, this field will be left empty. This field is not imported.

- *Field11: DB_Object_Synonym*
  Alternative gene symbol(s) or UniProtKB identifiers are provided pipe-separated, if available from UniProtKB. If none of these identifiers have been supplied, the field will be left empty. This field is not imported.

- *Field12: DB_Object_Type*
  The entity being annotated, that is always 'protein' for these files. This field is not imported.

- *Field13: Taxon*
  Identifier for the species. An interacting taxonomy ID may be included a pipe to separate it from the primary taxonomy ID. Examples: *taxon:9606* (human).

- *Field14: Date*
  The date of last annotation update in the format 'YYYYMMDD'.

- *Field15: Assigned_By*
  Attribution for the source of the annotation. Examples: *UniProtKB*, *AgBase*.

- *Field16: Annotation_Extension*
  This column contains cross-references to other ontologies/databases that can be

used to qualify or enhance the GO term applied in the annotation. This field is not imported.

- *Field17: Gene_Product_Form_ID*
  The identifier of a specific splice form of the *DB_Object_ID*. This field is not imported.

An example of record in *gene_association.goa_human* file is shown in Figure 58.

```
UniProtKB A0AV96   RBM47   GO:0005634   GO_REF:0000039   IEA   UniProtKB-SubCell:SL-0191   C
RNA-binding protein 47   RBM47_HUMAN|RBM47   protein   taxon:9606   20140607  UniProt
```

Figure 58 - Example of record in file gene_association.goa_human

In GOAAssociationLoader there is a unique method that manages the population of entries, containing only those field for tables *protein2biological_function_feature_imported* and *pub_ref_4_protein2biological_function_feature_imported*. This method is shown in Figure 59.

```
private void insertAssociation() {
   Reference source_id = null;
   if(!dbxref.isEmpty()){ //secondary association table
      for(Pair<String,String> p: dbxref)
         if(evidenceRefId != null && evidenceRefSourceName != null && evidence != null){
           assSecParams.get(PUBREF4PROT).get(PUBREF_EVIDENCE).setValue(evidence);
           assSecParams.get(PUBREF4PROT).get(PUBREF_QUALIFIER).setValue(qualifier_value);
           assSecParams.get(PUBREF4PROT).get(PUBREF_EV_REF_ID).setValue(evidenceRefId);
           assSecParams.get(PUBREF4PROT).get(PUBREF_EV_REF_SRC_NAME).setValue(evRefSourceName;
           String tempSourceName = evidenceRefSourceName.toUpperCase();
           if(tempSourceName.equals("REACTOME"))
              source_id = ReferenceDB.getInstance().getReferenceId("reactome");
           else
              if(tempSourceName.equals("PMID"))
                 source_id = ReferenceDB.getInstance().getReferenceId("pubmed");
              else
              if(tempSourceName.equals("GO_REF"))
                 source_id = ReferenceDB.getInstance().getReferenceId("goa");
             else
                 source_id = null;
            assSecParams.get(PUBREF4PROT).get(PUBREF_EV_REF_SOURCE).setValue(source_id);
            assSecParams.get(PUBREF4PROT).get(PUBREF_DATE).setValue(date);
            assSecParams.get(PUBREF4PROT).get(PUBREF_SUPP_ID).setValue(p.getSecond());
            assSecParams.get(PUBREF4PROT).get(PUBREF_SUPP_SOURCE_NAME).setValue(p.getFirst());
            assSecParams.get(PUBREF4PROT).get(PUBREF_REF_FILE).setValue(refFile);
            assSecEntries.get(PROT2BFF).get(PUBREF4PROT).
                 add(new AssociationSecondaryEntry(assSecParams.get(PUBREF4PROT), refFile));
            pubRefCnt++;
            if (pubRefCnt % BATCH_SIZE == 0)
               logger.debug("Inserted: " + pubRefCnt + " entries");
         }
      dbxref.clear();
   }
   assParams.get(PROT2BFF).get(PROT2BFF_ASSIGNED_BY).setValue(assigned_by);
   assParams.get(PROT2BFF).get(PROT2BFF_PROT_ID_VER).setValue(proteinId_version);
   assParams.get(PROT2BFF).get(PROT2BFF_INTER_WITH_TAX_ID).setValue(inter_with_tax_id);
   assParams.get(PROT2BFF).get(PROT2BFF_TAXONOMY_ID).setValue(taxonomy_id);
   assEntries.get(PROT2BFF).add(new AssociationEntry(proteinId, goId, "ANNOTATED_TO",
                                assParams.get(PROT2BFF), assSecEntries.get(PROT2BFF)));
   protein2bffCnt++;
   if (protein2bffCnt % BATCH_SIZE == 0)
      logger.debug("Inserted: " + protein2bffCnt + " entries");

}
```

Figure 59 - entry creation for GOA association tables

7.3 Post-processing and data recovery operations on import omim.txt data file

Before proceeding with the import of OMIM source files, the specific Importer *OmimImporter* must perform some pre-processing operations. In particular, it is necessary to normalize the information about phenotypes localization, that will be stored in the table *omim_clinical_synopsis*, in order to build a hierarchical structure of controlled terms. This requirement is due to the presence of several terms that define the same concept in OMIM free text files. This issue has been overcame by creating a supporting structure in *GPDW_definition.xml* where, for each possible value of field *name* in *omim_clinical_synopsis* table, preferred value, synonyms, group and hierarchical father values are defined. The skeleton of this supporting structure is displayed in Figure 60.

```
<omim_clinical_synopsis_synonyms source_handle="omim" file_handle="...">
        <group name="..." group_handle="..." />
        <group name="..." group_handle="..." />
        <item name="...">
           <synonym_name />
           <group handle="..."/>
        </item>
        <item name="...">
           <is_preferred />
           <group handle="..."/>
           <group handle="..."/>
        </item>
</omim_clinical_synopsis_synonyms>
```

Figure 60 - Base xml structure for OMIM clinical synopsis normalization

The values extracted from this supporting structure are inserted in the table *log.omim_clinical_synopsis_normalization*, that is queried by *OmimLoader* during the data extraction from file *omim.txt*.

The import phase for OMIM has to start with files *omim.txt* and *genemap.key* because they respectively provide information required to identify OMIM entries typology and the description of codes used in files *genemap* e *morbidmap*.

The flat file *omim.txt* contains the whole free text OMIM database. In this file, a record is identified by keyword **RECORD** and its fields are associated to keyword **FIELD* XX*, where *XX* is the label that identifies each field; the values that can be taken by this label are summarized in Table 6. The structure of an element *FIELD* may differ from that of other fields according to the semantic meaning of the information it contains; the number and the order of sub-fields is independently managed by the main field itself. The string **THE END** represents the end of the file.

Field label	Field name	Field cardinality
***FIELD* NO**	Mim number	Required
***FIELD* TI**	Title	Required
***FIELD* TX**	Text	Required
***FIELD* AV**	Allelic Variants	Optional in gene records. Not present in phenotype records
***FIELD* SA**	See also	Optional
***FIELD* RF**	References	Optional
***FIELD* CN**	Contributors	Optional. This label may be present two times
***FIELD* CD**	Creation Date	Required. This label may be present two times
***FIELD* ED**	Edit History	Optional. This label may be present two times
***FIELD* CS**	Clinical Synopsis	Optional

Table 6 - Field labels in file omim.txt

Below, a detailed explanation of the content of records is provided; it is essential for understanding the complex data format of the file that causes the modification of the standard procedures and the use of additional operations:

- **FIELD* NO*
  The value of *Mim number,* the unique identifier of OMIM entities, i.e. gene and genetic disorder, is extracted from this field.
- **FIELD* TI*
  This field is composed by the following sub-fields:
  - *Record type*. It is identified by the following symbols:
    - * : gene with known sequence;
    - +: gene with known sequence and phenotype;
    - #: phenotype with known molecular bases;
    - %: Mendelian phenotype with unknown molecular base;
    - ^ : history record;
    - The absence of symbol means phenotype with suspected Mendelian base.
  - *Mim number*. The unique identifier already extracted from previous field.
  - *Title*. Feature's title. In case of history record, the title is substituted by the operation performed:
    - *^275600 REMOVED FROM DATABASE* indicates that Mim id 275600 has been deleted from the database;
    - *^275650 MOVED TO 214950* indicates that the feature with id 275650 has been replaced by that with Mim number 214950.

- *Symbol (optional)*. The symbol of gene or genetic disorder.
- *Alternative title (optional)*. Text enclosed between sequence of characters “;;” and “;;” or between “;;” and “;”
- *Alternative symbol (optional)*. Text enclosed between sequence of characters “;” and “;;”.

- **FIELD* TX*
  The field is composed by three optional sub-fields: *Title*, *Subtitle* and *Body*.
- **FIELD* AV*
  The field is associated only to genes and provides notes on the allelic variants, i.e. the alteration of the normal gene sequence.
- **FIELD* SA*
  The field contains information about publication references but is not imported.
- **FIELD* RF*
  This optional field provides cross-references between OMIM record and scientific publications. Each reference is fully described with author(s), article’s title, book or journal’s name, year, volume’s number and pages.
- **FIELD* CN*
  Optional field used to insert references to contributors of OMIM database integration. This field includes the name of the author and the type of action, that can assume the following values: *updated*, *reorganized*, *revised*, *edited*, *reviewed*.
- **FIELD* CD*
  It contains references to the creation of the present record. The name of the author and the creation date of the current item are included in a single text line.
- **FIELD* ED*
  This optional field provides references to the revision of the present item. The field includes the author and the date of the review.
- **FIELD* CS*
  Optional field that contains data about phenotypes localization. This information contributes to the definition of the clinical synopsis structure described before. The field is divided into three sub-fields: *Title*, *Subtitle* and *Symptom*. This field may include one or many fields of type **FIELD* CD*, **FIELD* ED* and **FIELD* CN*.

All the clinical synopsis original names extracted from fields **FIELD* CS* populate the table *log.omim_clinical_synopsis_orig*, in order to keep trace of them. Moreover, preferred values of the same clinical synopsis group populate table *omim_clinical_synopsis_subgroup*.

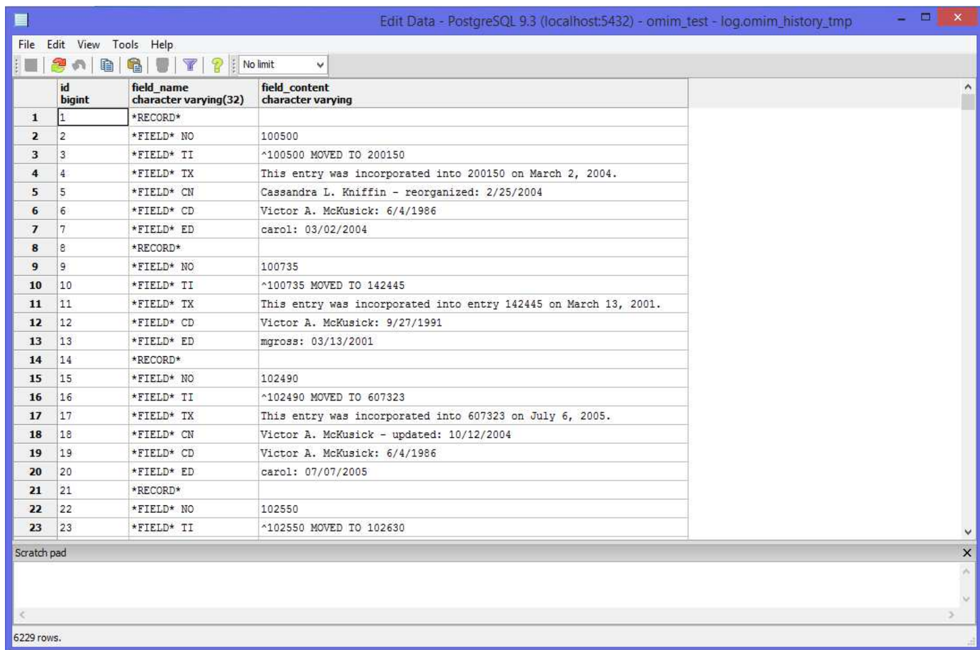
If an unknown value of clinical synopsis name is provided by *omim.txt*, in case of the release of a new version of the file, the system log prints an error message but the entry is still inserted in *omim_clinical_synopsys* table. The ontological structure of clinical synopsis created by pre-processing operations allows to fill the table *omim_clinical_synopsis_relationship* by inserting parent-child relations among the phenotype localization entities.
According to the record type, different source tables are populated by parsing *omim.txt*.

- If the record describes a gene:
  - *omim_gene;*
  - *omim_gene_alternative;*
  - *omim_gene_text;*
  - *omim_gene_edit_history;*
  - *omim_gene_contributors;*
  - *omim_gene_reference_publication;*
  - *omim_gene_allelic_variant;*
  - *omim_gene_clinical_synopsis_edit_history;*
  - *omim_gene_clinical_synopsis_contributors;*
- For record of phenotypes:
  - *omim_disorder;*
  - *omim_disorder_alternative;*
  - *omim_disorder_text;*
  - *omim_disorder_edit_history;*
  - *omim_disorder_contributor;*
  - *omim_disorder_reference_publication;*
  - *omim_disorder_clinical_synopsis_edit_history;*
  - *omim_disorder_clinical_synopsis_contributors;*
- Independently from record type, the following tables are populated when the field *FIELD* CS is present:
  - *omim_clinical_synopsis;*
  - *omim_clinical_synopsis_subgroup;*
  - *omim_clinical_synopsis_relationship.*

The population of history, relationship and association tables for gene and genetic disorder cannot be directly done in the moment in which the record is parsed.
This issue is due to the way how information is described in the source file and to the decision, when the extension of the automatic import procedures was designed, of realizing the entries' processing for all the imported tables in parallel.

Moreover, the population of source feature tables is done using the supporting TableLoader object, avoiding the direct access to the tables, unlike it was previously done, until the connection is closed and all the objects that access the DB are flushed.
In details, parsing and loading operations for *omim.txt* produce the following scenario:
during its execution, when the parser faces an historical record, that is recognized by the symbol '^' in **FIELD* TI,* it is not able to understand if the current entry refers to a gene or a phenotype, even if in the sub-field *Title* it is specified the feature that have replaced it in the source databank. There is no possibility of inserting the history entry in the proper table and, more important, all the data included in the other fields of the record, that may be still current and useful, are lost.
This happens because the information provided by the file is not sufficient to support a deeper analysis and because querying the database is avoided during importing, even useless when the replacing entity has not been parsed and imported yet.
Similar problems are encountered in populating relationship table *omim_gene_relationship* and *omim_disorder_relationship* and association table *gene2genetic_disorder_imported* when the information contained in the record is not sufficient to distinguish between gene and genetic disorders entities.
The solution to this problem was found by implementing a concrete instance of the post-processing module. This class, called *OmimLoaderPostProc.java*, was designed to execute its operation at Loader level. It was a necessary step because, as already said, the information provided by *omim.txt* is used to recognize data in the other OMIM files.
The workflow of the cooperation between OmimLoader and OmimLoaderPostProc can be summarized by the following points:

- If post-processing flag is off, history records parsed by OmimLoader are inserted in the database table *log.omim_history_tmp* as they are stored in the source file, by copying the full text of the entry. This trick allows to postpone the parsing of these data during the next post-processing phase. The content of *log.omim_history_tmp* is shown in Figure 61.



Edit Data - PostgreSQL 9.3 (localhost:5432) - omim_test - log.omim_history_tmp

File Edit View Tools Help

No limit

	id bigint	field_name character varying(32)	field_content character varying
1	1	*RECORD*	
2	2	*FIELD* NO	100500
3	3	*FIELD* TI	^100500 MOVED TO 200150
4	4	*FIELD* TX	This entry was incorporated into 200150 on March 2, 2004.
5	5	*FIELD* CN	Cassandra L. Kniffin - reorganized: 2/25/2004
6	6	*FIELD* CD	Victor A. McKusick: 6/4/1986
7	7	*FIELD* ED	carol: 03/02/2004
8	8	*RECORD*	
9	9	*FIELD* NO	100735
10	10	*FIELD* TI	^100735 MOVED TO 142445
11	11	*FIELD* TX	This entry was incorporated into entry 142445 on March 13, 2001.
12	12	*FIELD* CD	Victor A. McKusick: 9/27/1991
13	13	*FIELD* ED	mgross: 03/13/2001
14	14	*RECORD*	
15	15	*FIELD* NO	102490
16	16	*FIELD* TI	^102490 MOVED TO 607323
17	17	*FIELD* TX	This entry was incorporated into 607323 on July 6, 2005.
18	18	*FIELD* CN	Victor A. McKusick - updated: 10/12/2004
19	19	*FIELD* CD	Victor A. McKusick: 6/4/1986
20	20	*FIELD* ED	carol: 07/07/2005
21	21	*RECORD*	
22	22	*FIELD* NO	102550
23	23	*FIELD* TI	^102550 MOVED TO 102630

Scratch pad

6229 rows.

Figure 61 - log.omim_history_tmp table

- When, for a certain record of gene or disorder, the origin of the Mim number associated to the current one cannot be identified, the information is parsed and inserted into a specific ArrayList, where it is stored until the end of file is reached. Before disposing the Loader, these data populate one of the temporary tables *log.omim_gene_association_tmp* and *log.omim_disorder_association_tmp*, specifically designed to prepare the population of the final relationship and association tables.
- After that OmimLoader is disposed, post-processing class, defined in xml configuration file, component is configured. It creates a new instance of OmimLoader passing as input stream the content of table *log.omim_history_tmp*. Post-processing flag is enabled and the standard operations of parsing and loading of data are performed. A warning message is sent to the log for those records marked that cannot be inserted in the proper history table in any case, as shown in Figure 62.

[2014-06-19 08:01:59,303]TRACE[omim] - OmimLoader - History record with mimID: 102930 is REMOVED FROM DATABASE. This entry is not inserted into the DB.

[2014-06-19 08:01:59,303]TRACE[omim] - OmimLoader - History record with mimID: 194530 is REMOVED FROM DATABASE. This entry is not inserted into the DB.

[2014-06-19 07:59:22,447]TRACE[omim] - OmimLoader - History record with mimID: 600199 is MOVED TO mimID: 300021 that is unknown. This entry is not inserted into the DB.

[2014-06-19 07:59:22,585]TRACE[omim] - OmimLoader - History record with mimID: 600255 is MOVED TO mimID: 400000 that is unknown. This entry is not inserted into the DB.

Figure 62 - System log warning messages for omim.txt

- Finally, when all the data of *omim_gene* and *omim_disorder* tables have been loaded in the database, the SQL queries displayed in figures 63 and 64 are executed in order to insert data stored in temporary tables in the proper association and relationship tables.

```
INSERT INTO omim_gene_relationship
(SELECT DISTINCT a.omim_gene_oid, b.omim_gene_oid, tmp.relationship_type, tmp.inferred,
tmp.reference_file
FROM log.omim_gene_associations_tmp AS tmp
JOIN omim_gene AS a ON tmp.source1_id=a.omim_gene_id AND
tmp.source1_name=a.omim_gene_source
JOIN omim_gene AS b ON tmp.source2_id=b.omim_gene_id AND
tmp.source2_name=b.omim_gene_source
WHERE (a.omim_gene_oid, b.omim_gene_oid, tmp.relationship_type, tmp.inferred,
tmp.reference_file)
NOT IN (SELECT term_oid, related_term_oid, relationship_type, inferred, reference_file
FROM omim_gene_relationship))

INSERT INTO omim_disorder_relationship
(SELECT DISTINCT a.omim_disorder_oid, b.omim_disorde_oid, tmp.relationship_type,
tmp.inferred, tmp.reference_file
FROM log.omim_disorder_associations_tmp AS tmp
JOIN omim_disorder AS a ON tmp.source1_id=a.omim_disorder_id AND
tmp.source1_name=a.omim_disorder_source
JOIN omim_disorder AS b ON tmp.source2_id=b.disorder_gene_id AND
tmp.source2_name=b.omim_disorder_source
WHERE (a.omim_disorder_oid, b.omim_disorder_oid, tmp.relationship_type, tmp.inferred,
tmp.reference_file)
NOT IN (SELECT term_oid, related_term_oid, relationship_type, inferred, reference_file
FROM omim_disorder_relationship))
```

Figure 63 - SQL query for OMIM relationship tables

```
INSERT INTO gene2genetic_disorder_imported
(SELECT nextval('value'), source1_id, source1_name, source2_id, source2_name, reference,
reference_file,  association_type, taxonomy_id
FROM log.omim_gene_associations_tmp
WHERE (source2_id, source2_name) IN
(SELECT omim_disorder_id, omim_disorder_source
FROM omim_disorder
WHERE omim_disorder_id=source2_id AND omim_disorder_source=source2_name))

INSERT INTO gene2genetic_disorder_imported
(SELECT nextval('value'), source1_id, source1_name, source2_id, source2_name, reference,
reference_file,  association_type, taxonomy_id
FROM log.omim_disorder_associations_tmp
WHERE (source2_id, source2_name) IN
(SELECT omim_gene_id, omim_disorder_source
FROM omim_gene
WHERE omim_gene_id=source2_id AND omim_gene_source=source2_name))

nextval('value') retrieves the last oid in the databse, after that the ownership of the generator is passed to the DBMS.
Finally the ownership is reclaimed and the OidGenerator is synchronized with the state of the sequence on the DBMS.
```

Figure 64 - SQL query for gene2genetic_disorder_imported table

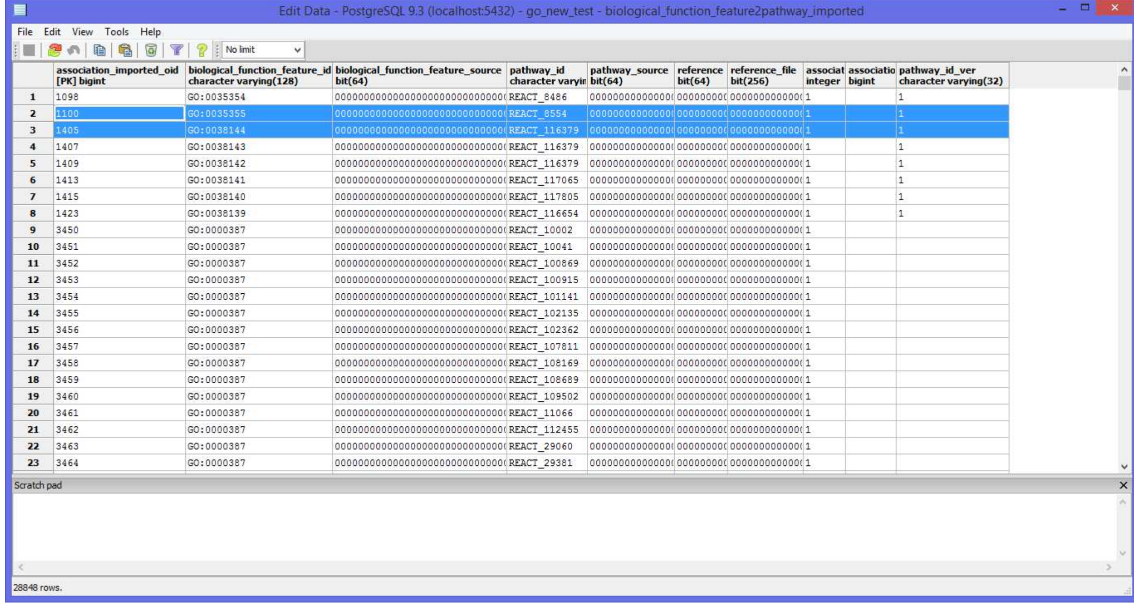
8. Validation and testing

8.1 Imported data errors and inconsistencies

This section groups all the anomalies found during the import operations of the considered data source files. Additional testing has been performed to validate the content of data in imported tables and new technical solutions have been proposed to manage errors and inconsistencies.

8.1.1 GO

The anomalies encountered during the import of file *go_daily-termdb.obo-xml* concern the definition and the content of <xref_analog> and <synonym> tags in many GO term description.
The element <xref_analog> is used to specify id and name of entity from different data sources that are referred by the current term to describes associations.
A limited number of pathway identifiers from REACTOME do not match the regular expressions provided for the databank; this particular format, indeed, includes details on the pathaway id version. Using string management's methods, pathway ids are formatted in the proper manner. In order to store this additional information, the definition of *biological_function_feature2pathway_imported* table has been updated by adding the column *pathway_id_ver* at the end, as shown in Figure 65.



Edit Data - PostgreSQL 9.3 (localhost:5432) - go_new_test - biological_function_feature2pathway_imported

File Edit View Tools Help

No limit

	association_imported_oid [PK] bigint	biological_function_feature_id character varying(128)	biological_function_feature_source bit(64)	pathway_id character varyir	pathway_source bit(64)	reference bit(64)	reference_file bit(256)	associat integer	associatio bigint	pathway_id_ver character varying(32)
1	1098	GO:0035354	00000000000000000000000000000(	REACT_8486	00000000000000(	00000000(	000000000000(	1		1
2	1100	GO:0035355	00000000000000000000000000000(	REACT_8554	00000000000000(	00000000(	000000000000(	1		1
3	1405	GO:0038144	00000000000000000000000000000(	REACT_116379	00000000000000(	00000000(	000000000000(	1		1
4	1407	GO:0038143	00000000000000000000000000000(	REACT_116379	00000000000000(	00000000(	000000000000(	1		1
5	1409	GO:0038142	00000000000000000000000000000(	REACT_116379	00000000000000(	00000000(	000000000000(	1		1
6	1413	GO:0038141	00000000000000000000000000000(	REACT_117065	00000000000000(	00000000(	000000000000(	1		1
7	1415	GO:0038140	00000000000000000000000000000(	REACT_117805	00000000000000(	00000000(	000000000000(	1		1
8	1423	GO:0038139	00000000000000000000000000000(	REACT_116654	00000000000000(	00000000(	000000000000(	1		1
9	3450	GO:0000387	00000000000000000000000000000(	REACT_10002	00000000000000(	00000000(	000000000000(	1		
10	3451	GO:0000387	00000000000000000000000000000(	REACT_10041	00000000000000(	00000000(	000000000000(	1		
11	3452	GO:0000387	00000000000000000000000000000(	REACT_100869	00000000000000(	00000000(	000000000000(	1		
12	3453	GO:0000387	00000000000000000000000000000(	REACT_100915	00000000000000(	00000000(	000000000000(	1		
13	3454	GO:0000387	00000000000000000000000000000(	REACT_101141	00000000000000(	00000000(	000000000000(	1		
14	3455	GO:0000387	00000000000000000000000000000(	REACT_102135	00000000000000(	00000000(	000000000000(	1		
15	3456	GO:0000387	00000000000000000000000000000(	REACT_102362	00000000000000(	00000000(	000000000000(	1		
16	3457	GO:0000387	00000000000000000000000000000(	REACT_107811	00000000000000(	00000000(	000000000000(	1		
17	3458	GO:0000387	00000000000000000000000000000(	REACT_108169	00000000000000(	00000000(	000000000000(	1		
18	3459	GO:0000387	00000000000000000000000000000(	REACT_108689	00000000000000(	00000000(	000000000000(	1		
19	3460	GO:0000387	00000000000000000000000000000(	REACT_109502	00000000000000(	00000000(	000000000000(	1		
20	3461	GO:0000387	00000000000000000000000000000(	REACT_11066	00000000000000(	00000000(	000000000000(	1		
21	3462	GO:0000387	00000000000000000000000000000(	REACT_112455	00000000000000(	00000000(	000000000000(	1		
22	3463	GO:0000387	00000000000000000000000000000(	REACT_29060	00000000000000(	00000000(	000000000000(	1		
23	3464	GO:0000387	00000000000000000000000000000(	REACT_29381	00000000000000(	00000000(	000000000000(	1		

Scratch pad

28848 rows.

Figure 65 – biological_function_feature2pathway_imported table

The second anomaly in the source data file was encountered in the description of term with identifier *GO:0031362* where, for a given synonym definition, null value of tag <synonym_text> and empty value of attribute *scope* generate an *IllegalArgumentException* error message in the log that forces the termination of the parser.
This issue has been solved by adding a conditional block to avoid stopping the execution of the parsing and loading of the obo-xml file. In this case, a warning message is printed in the system log, specifying the details of the error.

8.1.2 GOA

No anomalies have been discovered during the import operations of files *ec2go*, *interpro2go*, *pfma2go* and *gp2protein.geneid*. Analysis and testing mainly focused on the huge amount of records imported from files *gene_association.goa_<species>*, already described in Chapter 7, section 2.2.
The complexity of the relations between the fields described in these files required a deep investigation of the information included in the UniProt-GOA website and in the README files provided there. Furthermore, a direct explaination request to database curators have been sent to confirm the analysis carried out.
In details, these files populate the table *protein2biological_function_feature_imported* and *pub_ref_4_protein2biological_function_feature_imported* shown in the logical diagram of GOA.
A set of tests have been performed to validate the proposed organization of the data into the two tables, ensuring that no additional tuples are generated and no related information are separated by this logical division. A temporary table including all the file's fields under analysis have been created to run the tests. Two examples of these tests are displayed in Figure 66: the former checks the cardinality of column *evidence_ref_id* (derived from file's field *DB:Reference*) for each association record; the latter verifies that columns *supporting_id* and *supporting_source_name* (derived from file's field *With (or) From*) are related to the data in *evidence_ref_id* and not to the association imported oid.

```
SELECT COUNT (DISTINCT evidence), association_imported_oid, evidence_ref_id, evidence_ref_source_name
FROM pub_ref_4_protein2bff_temp
GROUP BY association_imported_oid, evidence_ref_id, evidence_ref_source_name
HAVING COUNT (DISTINCT evidence) > 1

SELECT association_imported_oid, COUNT (DISTINCT evidenc_ref_id), COUNT (supporting_id)
FROM pub_ref_4_protein2bff_temp
GROUP BY association_imported_oid
HAVING COUNT (supporting_id) > 1 AND COUNT (evidence_ref_id) > 1
```

Figure 66 – SQL queries for GOA secondary association table

The result of the tests shows that a further organization of the data that populate *pub_ref_4_protein2biological_function_feature_imported* table is not allowed; all the fields that are object of the analysis have to be constrained by a unique index to ensure the goodness of imported data.

A new solution for the organization of secondary association tables has been proposed in order to simplify data access and speed-up querying time, that are necessary for densely populated tables. This proposal includes the addition of another level of secondary association tables but it has not been still implemented because it probably requires several changes in both AssocitionDataLoader and xml configuration files' structure.

The usage of this enhancement, that will be definitively implemented in the future developments of GPDW project, for GOA association tables is sketched in Figure 67.

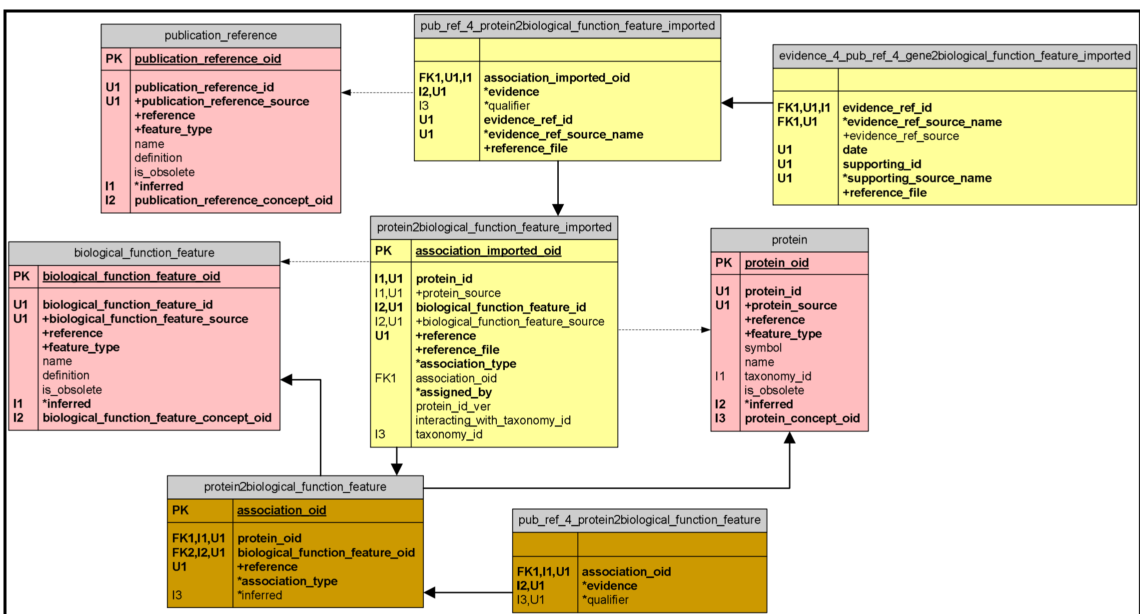


Figure 67 – Proposed solution for GOA secondary association tables

8.1.3 Entrez Gene

No anomalies or errors have been encountered during the import of association data between gene and biological_function_feature entities from the single file, *gene2go*, that has been considered in the development of the Thesis.

8.1.4 ExPASy ENZYME

There are no inconsistencies discovered during the import of files *enzclass.txt* and *enzyme.dat*. However it has been decided to report in the system log few unusual situations that can help the administrator to verify the absence of errors into the imported data:

- The import of EC numbers that identify enzymes for which there exists no enzyme class or subclass they belong to.
- In *enzyme.dat*, the matching of substrings in field *Comment*, labeled with symbol CC, with the recognized similarity patterns that describe the enzyme actions. In few cases no related EC numbers are not specified and similarity relations cannot be identified.

8.1.5 OMIM

Below there is an overview of the problems found during the validation of the import procedures for OMIM files. The action taken to solve these issues are briefly described.

- *omim.txt*
  - During the parsing of field *FIELD* CS several errors due to a wrong format of the textual block describing the clinical synopsis information have been discovered; the anomalies are reported in the log of the program. In case of new definitions of clinical synopsis in the new release of the file, the administrator can update the *omim_clinical_synopsis_synonyn* structure to overcome the errors.
  - The records in the field *FIELD* CN where there are not present the name and action of the contributor to the integration of OMIM database are discarded and the incosistent information is reported in the log.
  - As shown in Figure 62, warning messages are sent to the log of the program for history records that, after the post-processing operations, cannot be inserted in the correct history table because of the uncertainty on the feature type. This issue is reflected and may propagates to the parsing of *genemap* and *morbidmap* files.
- *genemap*
  - According to the content of *genemap.key* file, the records provided by *genemap* are made of 18 fields. In many cases, when the last field *Reference* is not specified, the record counts only seventeen columns and the TabularFileParser may generate an exception. This problem goes over through redefining the method that check this event so that no warning messages are displayed.

- A list, made of 11 items, of the method codes that differ from the expected flag values defined in *genemap.key* is returned at the end of GenemapLoader execution. These entries are not inserted in the database.
- From the set of fields named *Disorders* it is possible to retrieve the name and the Mim number of the phenotypes related to the current entity, that can be a gene or another disorder. There were discovered more than 700 records that contain only the name of disorder; the lack of the identifiers determines the impossibility of inserting these additional information in the database, hence data from this field are discarded and warning messages are displayed in the log.

8.2 Quantification of imported data and running time

Below there is a list of tables containing running time and quantification of imported data from the data sources considered for the development of the Thesis. Testing phase has been performed on a machine with processor Intel Core™ i5-3210M with 2.5Ghz processor speed and disk HITACHI with 5400RPM and 8Mb of cache.

Data source	Parser	Table populated	Number of entries	Running time
GO	GOLoader	22	328.202	223 sec
GOA	EcToGoLoader	1	5.253	1 sec
	PfamToGoLoader	1	10.579	2 sec
	InterproToGoLoader	1	28.393	6 sec
	GpToProteinLoader	1	11.547.100	2.471 sec
	GOAAssociationLoader	2	597.351.000	113.550 sec
Entrez Gene	GeneToGoLoader	2	1.595.794	315 sec
ExPASy ENZYME	EnzymeClassLoader	2	660	< 1 sec
	EnzymeDataLoader	9	38.506	12 sec
OMIM	OmimLoader	22	818.190	8.095 sec
	GenemapKeyLoader	0	-	< 1 sec
	GenemapLaoder	10	51.779	662 sec
	MorbidmapLoader	6	12.358	88 sec
	PubmedLoader	2	143.745	2.131 sec

Table 7 – Total number of imported entries and running time

Data source	Table name	Number of entries
GO	biological_function_feature_similarity_imported	1.660
	biological_function_feature_history_imported	2.225
	biological_function_feature2enzyme_imported	5.407
	biological_function_feature2pathway_imported	28.848
	go_biological_process	25.390
	go_biological_process_synonym	58.633
	go_biological_process_subset	5.772
	go_biological_process_dbxref	1.698
	go_biological_process_definition_dbxref	44.612
	go_biological_process_relationship	56.533
	go_cellular_component	3.298
	go_cellular_component_synonym	2.919
	go_cellular_component_subset	829
	go_cellular_component_dbxref	406
	go_cellular_component_definition_dbxref	5.952
	go_cellular_component_relationship	5.868
	go_molecular_function	10.446
	go_molecular_function_synonym	26.502
	go_molecular_function_subset	3.264
	go_molecular_dbxref	9.795
	go_molecular_function_definition_dbxref	16.651
	go_molecular_function_relationship	11.494

Table 8 – Details of imported tables of GO

Data source	Table name	Number of entries
GOA	biological_function_feature_imported2enzyme_imported	5.271
	biological_function_feature_imported2protein_fam_dom_imported	38.972
	gene2protein_imported	11.547.100
	protein2biological_function_feature_imported	264.694.000
	pub_ref_4_protein2biological_function_feature_imported	332.657.000

Table 9 – Details of imported tables of GOA

Data source	Table name	Number of entries
Entrez Gene	gene2biological_function_feature_imported	1.171.140
	pub_ref_4_gene2biological_function_feature_imported	424.654

Table 10 – Details of imported tables of Entrez Gene

Data source	Table name	Number of entries
ExPASy ENZYME	enzyme2protein_fam_dom_imported	1.283
	enzyme_history_imported	2150
	enzyme_similarity_imported	48
	expasy_enzyme	5.778
	expasy_enzyme_action	872
	expasy_enzyme_alternative_name	9.326
	expasy_enzyme_comment	9.940
	expasy_enzyme_relationship	5.772
	protein2enzyme_imported	3.997

Table 11 – Details of imported tables of ExPASy ENZYME

Data source	Table name	Number of entries
OMIM	gene2clinical_synopsis_imported	952
	gene2genetic_disorder_imported	6.341
	gene2publication_reference_imported	91.061
	gene_history_imported	473
	genetic_disorder2clinical_synopsis_imported	25.556
	genetic_disorder2publication_reference_imported	52.480
	genetic_disorder_history_imported	394
	omim_clinical_synopsis	77
	omim_clinical_synopsis_relationship	19
	omim_clinical_synopsis_subgroup	78
	omim_disorder	5.018
	omim_disorder_alternative	4.604
	omim_disorder_clinical_synopsis_contributors	956
	omim_disorder_clinical_synopsis_edit_history	5.438
	omim_disorder_contributors	7.206
	omim_disorder_edit_history	41.996
	omim_disorder_method	617
	omim_disorder_reference_publication	30.700
	omim_disorder_relationship	3.203
	omim_disorder_text	11.060
	omim_gene	10.324
	omim_gene_allelic_variant	17.634
	omim_gene_alternative	11.123
	omim_gene_clinical_synopsis_contributors	35
	omim_gene_clinical_synopsis_edit_history	66
	omim_gene_contributors	29.869
	omim_gene_edit_history	96.239
	omim_gene_method	12.611
	omim_gene_reference_publication	76.274
	omim_gene_relationship	32.935
	omim_gene_text	38.542
	phenotype_4_gene2clinical_synopsis_imported	6.534
	phenotype_4_genetic_disorder2clinical_synopsis_imported	185.089

Table 12 – Details of imported tables of OMIM

9. Final discussions and conclusions

The proposed goals of this Thesis have been achieved through a reengineering activity of the GPDW framework, the renovation of data importing procedures, the enhancements of configuration processes and the creation of new software packages that accomplish the requirement of modularity, abstraction and generalization of the automatic procedures of GPDW data import and integration.

The final result of this Thesis is represented by the integration of new components, described in Chapter 5, to the existing software framework; they generalize and ease the automatic import, aggregation and quality improvement of genomic and proteomic data from different biomolecular databanks, which will be available to the scientific community through the Genomic and Proteomic Knowledge Base (GPKB) Web interface, that is available from http://www.bioinformatics.dei.polimi.it/GPKB/ [29].

The work has been scheduled into three main stages:

1. Analysis of the existing GPDW framework in order to highlight design limitations, limitations and procedures that needed to be optimized.
2. Design and implementation of generic and customizable procedures for automatic importing and integration of data in the data warehouse. This stage can be divided into three main activities:
    - Enhancements of *GPDW_definition.xml* and *feature_definitions.xml* configuration files in order to overcome limitations in the description of features and sources and to introduce new templates for the definition of feature tables.
    - Development of generic and reusable Loader components for the automatic import of source feature tables.
    - Design of new modules for post-processing and data recovery operations that can be implemented at different stages of the import process.
3. Application of the implemented procedures for the importing of data provided by Gene Ontology, GOA, Entrez Gene, OMIM and ExPASy ENZYME databanks. In particular, this task required the following activities:
    - Analysis of source data files to verify the consistency and the truthfulness of conceptual and logical diagrams previously designed for the considered data sources; the update of these schemas to face changes in data provided by the considered databanks was performed, when necessary.
    - Insertion in the GPDW framework configuration files of source definitions and of the description of data file content, i.e. feature and annotation items.

When additional information are available, directly or indirectly, from the data, the description is extended with data of similarity, history, hierarchical relationship, sub-features and external references to information supplied by other sources.

- Development of Parser and specific Loader java objects, by extending the developed generic classes, to extract data from the considered sources and enabling their automatic importing in the data warehouse.
- Validation of the generic procedures for specific source cases. This step can highlight data errors, also due to particular data of the considered source, that may be fixed through the use of specific post-processing and data recovery operations, whose generic calling has been designed and implemented.
- Final testing and validation of imported data, quantification of errors and inconsistencies, and evaluation of import time.

The use of generic abstract classes, the implementation of new automatic procedures for the import of source tables, and the design and development of generic classes for data recovery and post-processing make the import layer completely generalized and available for considering a growing number of databanks. The developed importing architecture allows a robust control of imported data and their quality and provenance, and opens the way to validate their high accuracy by cross-references between data imported from different sources.

The amount and, mainly, the quality of data concerning both biological and biomolecular entities that have been imported and integrated in the data warehouse may unveil new knowledge about genomice and proteomice features. The quality check automatically performed in parsing, data loading and at the end of the import process may also reveal inconsistencies and missing information in the databanks, supporting the quality improvement of genomic and proteomic data available and used by biologists and researchers.

10. Future developments

Genomic and Proteomic Data Warehouse project aims to introduce a novel approach to support the integration of a considerable number of data sources that deal with biomedical and biomolecular information.

GPDW project will continue by increasing the number of considered data sources. The import approach, developed with the contribution of the enhancements described in this Thesis, that is now applied to Gene Ontology, GOA, EntrezGene, GAD, OMIM and ExPASy ENZYME will be extended to other public databanks that are matter of interest for the goals of the project.

A future development concerns update of the conceptual and logical design of database schemas. The designed diagrams derive from the concrete translation for the specific sources of the global conceptual data model. The objective will be to extend the model, preserving its key feature, abstraction, generalization and customization, in order to improve the logical organization of data. For example, the introduction a second level of additional association tables and additional feature tables may reduce the complexity of database tables and speed-up their access time.

This task will require a partial reimplementation of Loader java objects that manage this specific table's types and, moreover, the review of XML configuration files to enable the extension of the automatic import procedures to the new database tables.

Furthermore, the implementation of procedures for the aggregation and elimination of replicated records is still problematic and time-consuming; a simplification of the whole replicates checking and merging/elimination process can be useful to reduce its impact in terms of time and memory waste, in particular when the amount of file entries is quite large.

Finally, it is already planned the update of the group of packages for downloading the files of the considered databanks directly from the original source, in order to fully automate the import process, supply frequently updated information to the framework and, consequently, guarantee higher quality and consistency of imported data.

References

[1] Masseroli M. Biomolecular Databanks. Bioinformatica e biologia computazionale per la medicina molecolare [Online]; 2010. Available from: http://www.bioinformatics.polimi.it/masseroli/bbcmm/dispense/9_BiomolecularDataBanks.pdf

[2] Canakoglu A. Integration of biomolecular interaction data in a genomic and proteomic data warehouse [Online]; 2011; p. 5-6, 20-24. Available from: https://www.politesi.polimi.it/handle/10589/16924

[3] Wilkins MR, Pasquali C, Appel RD, Ou K, Golaz O, Sanchez JC et al. From proteins to proteomes: large scale protein identification by two-dimensional electrophoresis and amino acid analysis. *Bio/technology (Nature Publishing Company)*; 1996; 14(1):61-65.

[4] Masseroli M. BioMedical Terminologies and Ontologies. ICT for Health Care and Life Sciences [Online]; 2013. Available from: http://www.bioinformatics.deib.polimi.it/masseroli/ICT4HCLS/material/4a_BioMedicalTerminologies-Ontologies.pdf

[5] Masseroli M, Pinciroli F. Using Gene Ontology and genomic controlled vocabularies to analyze high-throughput gene lists: three tool comparison. *Computer in biology and medicine*. 2006 Jul-Aug; 36(7-8): p.731-747.

[6] Matthews L, Gopinath G, Gillespie M, Caudy M, Croft D, de Bono B et al. Reactome knowledgebase of human biological pathways and processes. *Nucleic Acids Res*. Jan 2009; 37:D619-D622.

[7] OBO - Open Biological and Biomedical Ontologies. 2014. Available from: http://www.obofoundry.org/about.shtml

[8] Smith B, Ashburner M, Rosse C, Bard J, Ceusters W, Goldberg LJ et al. The OBO Foundry: coordinated evolution of ontologies to support biomedical data integration. *Nat Biotechnol*. 2007 Nov; 25(11):1251-1255.

[9] Khatri P, Done B, Rao A, Done A, Draghici S. A semantic analysis of the annotations of the human genome. Bioinformatics 2005, 21(16):3416-3421.

[10]Masseroli M, Chicco D, Pinoli P. Probabilistic Latent Semantic Analysis for prediction of Gene Ontology annotations. In: Proc. WCCI 2012 IEEE World Congress on Computational Intelligence; The 2012 Int. Joint Conf. Neural Networks (IJCNN). Edited by Abbass HA. Piscataway, NJ, IEEE 2012, 2891-2898.

[11] Sharan R, Ulitsky I, Shamir R. Network-based prediction of protein function. *Mol. Syst. Biol.* 2007, 3:88.

[12] Masseroli M, Canakoglu A, Quigliatti M. Detection of Gene Annotations and Protein-Protein Interaction Associated Disorders through Transitive Relationships between Integrated Annotations. *BMC Bioinformatics*. 2014 (submitted for publication).

[13] Masseroli M, Canakoglu A. Genomic and proteomic data integration and identification of missing annotations. In: Fogolari F, Policriti A, editors. *BITS 2013: 10th Annual Meeting of the Bioinformatics Italian Society*; May 21-23, 2013; Udine, IT. Udine, IT: IGA; 2013. p. 101.

[14] Atzeni P, Ceri S, Fraternali P, Paraboschi S, Torlone R. Basi di dati: Architetture e linee di evoluzione. McGraw-Hill Italia. 2007.

[15] Maffezzoli A, Masseroli M. Chapter XLV: Genomic databanks for biomedical informatics. In: Lazakidou AA, editor. *Handbook of Research on Informatics in Healthcare and Biomedicine*. ISBN 1-59140-982-9. Hershey, PA: Idea Group Inc.; 2006. P.357-366.

[16] Ghisalberti G, Masseroli M. Tassonomia di file di dati di annotazioni genomiche e proteomiche. Bioinformatica e biologia computazionale per la medicina molecolare. [Online]; 2010. Available from: http://www.bioinformatics.polimi.it/masseroli/bbcmm/dispense/esercitazioni/E4_TassonomiaFileDati.pdf

[17] XML - eXtensible Markup Language. 2014.
Available from: http://www.w3.org/standards/xml/core

[18] Canakoglu A, Masseroli M. Genomic and proteomic data integration for comprehensive biodata search. In: Masseroli M, Romano P, Lisacek F, editors. *EMBnet.journal 2012 Nov; 18(Supplement B) NETTAB 2012: "Integrated Bio-Search"*; 2012 Nov 14-16; Como, IT. ISSN 1023-4144. Nijmegen, NL: EMBnet Stichting; 2012. Vol. 18, Supplement B, p. 89-91.

[19] Davidson SB, Overton C, Tanen V, Wong L. BioKleisli: a digital library for biomedical researchers. *Int. J. Digit. Libr.* 1997, 1:36-53.

[20] Davidson SB, Crabtree J et al. K2/Kleisli and GUS: Experiments in integrated access to genomic data sources. *IBM System Journal* vol. 40, no. 2, p. 512-531, 2001.

[21] Canakoglu A, Masseroli M, Ceri S, Tettamanti L, Ghisalberti G, Campi A. Integrative warehousing of biomolecular information to support complex multi-topic queries for biomedical knowledge discovery. In: Nikita SK, Fotiadis DI, editors. *Proceedings of the 2013 Thirteenth IEEE International Conference on Bioinformatics and Bioengineering: BIBE 2013,* 2013 Nov 10-13; Chania, GR. [CD-ROM] ISBN 978-1-4799-3163-7. Los Alamitos, CA: IEEE Computer Society; 2013. 159, p. 1-4.

[22] Canakoglu A, Ghisalberti G, Masseroli M. Integration of biomolecular interaction data in a genomic and proteomic data warehouse to support biomedical knowledge discovery. In: Biganzoli E, Vellido A, Ambrogi F, Tagliaferri R, editors. *Computational*

Intelligence Methods for Bioinformatics and Biostatistics. ISBN 978-3-642-35685-8. Heidelberg, D: Springer; 2012. p. 112-126. (Lecture Notes in Bioinformatics; vol 7548).

[23] Sujansky W. Heterogeneous database integration in biomedicine. *J. Biomed. Inform.* vol 34, no. 4, p. 285-298, 2001.

[24]PCRE - Perl Compatible Regular Expression. 2014. Available from: http://www.pcre.org/pcre.txt

[25] Di Stefano D. Architetture per l'importazione automatica di dati genomici e proteomici in un data warehouse integrato [Online]; 2011. p. 19-20, 37-43.
Available from: https://www.politesi.polimi.it/handle/10589/16983

[26] GO - Gene Ontology. 2014.
Available from: http://www.geneontology.org/GO.doc.shtml

[27] Harris M, Lomax J, Ireland A, Clark JI. The Gene Ontology project. *John Wiley & Sons, Ltd. Encyclopedia of Genetics, Genomics, Proteomics and Bioinformatics*. Part 4. Bioinformatics 4.7. Structuring and Integrating Data, 2005.

[28] Camon E, Magrane M, Barrell D, Lee V, Dimmer E, Maslen J et al. The Gene Ontology Annotation (GOA) Database: sharing knowledge in Uniprot with Gene Ontology. *Nucleic Acids Res.* Jan 1, 2004; 32: D262-D266.

[29] Maglott D, Ostell J, Tatusova T. Entrez Gene: gene-centered information at NCBI. *Nucleic Acids Res.* (2011) 39 (suppl 1): D52-D57.

[30] ExPASy ENZYME - Expert Protein Analysis System ENZYME. 2014. Available from: http://enzyme.expasy.org/

[31] Artimo P, Jonnlagedda M, Arnold K, Baratin D, Csardi G, de Castro E et al. ExPASy: SIB bioformatics resource portal. *Nucleic Acids Res.* 2012 Jul; 40:W597-603.

[32] OMIM - Online Mendelian Inheritance in Man. 2014. Available from: http://www.ncbi.nlm.nih.gov/omim

[33] Amberger J, Bocchini CA, Scott AF, Hamosh A. McKusick's Online Mendelian Inheritance in Man (OMIM). *Nucleic Acids Res.* 2009 Jan; 37: D793-6.

[34] van Vlie H. Software Engineering: Principles and Practice. 3rd ed. Wiley. 2008.

[35] Elmasri R, Navathe SB. Fundamentals of Database Systems. 6th ed. Pearson. 2010

[36] Document Object Model (DOM). 2014.

Available from: http://www.w3.org/DOM/

[37] Gene Ontology Reference Collection. 2012.
Available from: http://www.geneontology.org/doc/GO.references