Collaborative Knowledge Management in the Life Sciences Network

Inaugural – Dissertation

zur

Erlangung des Doktorgrades der

Mathematisch-Naturwissenschaftlichen Fakultät

der Heinrich-Heine-Universität Düsseldorf

vorgelegt von

Ingo Paulsen

aus Duisburg

Oktober 2007

Aus dem Institut für Informatik
der Heinrich-Heine-Universität Düsseldorf

Gedruckt mit der Genehmigung der Mathematisch-Naturwissenschaftlichen
Fakultät der Heinrich-Heine-Universität Düsseldorf

Referent: Prof. Dr. Arndt von Haeseler
Korreferent: Prof. Dr. Stefan Conrad

Tag der mündlichen Prüfung: 14.01.2008

Acknowledgments

I wish to thank my supervisor Arndt von Haeseler for his excellent advise, collaborations, and his friendly behaviour. Also I want to thank Stefan Conrad for accepting the task to read this thesis as a second reviewer.
Special thanks to Katrin, Indra, Dominic, and Jochen. Furthermore, I would like to thank Tanja, Andrea, Markus, Alex, Simone, Nahal, Thomas L., Nicole, Thomas S., Lutz, Claudia, Anja, and all other Ontoverse project partners and colleagues of the Bioinformatics Department in Düsseldorf.
I am grateful to my parents, my sister, my niece, my grandparents, and my aunt.
Financial support from the German Federal Ministry of Education and Research and the Deutsche Forschungsgemeinschaft is gratefully acknowledged.

Publications

Parts of this thesis have been published in the following articles and conference proceedings:

1. Ingo Paulsen, Dominic Mainz, Katrin Weller, Indra Mainz, Jochen Kohl, Arndt von Haeseler. (2007) ONTOVERSE: Collaborative Knowledge Management in the Life Sciences Network. In: *Proceedings of the Germany eScience Conference 2007*, Max Planck Digital Library, ID 316588.0.

2. Ingo Paulsen, Arndt von Haeseler. (2006) INVHOGEN: a database of homologous invertebrate genes. *Nucleic Acids Res.*, **34**, D349-D353.

Other publications:

1. Jochen Kohl, Ingo Paulsen, Thomas Laubach, Achim Radtke, Arndt von Haeseler. (2006) HVRBASE++: a phylogenetic database for primate species. *Nucleic Acids Res.*, **34**, D700-D704.

2. Katrin Weller, Dominic Mainz, Indra Mainz, Ingo Paulsen: Wissenschaft 2.0? Social Software im Einsatz für die Wissenschaft. In: Marlies Ockenfeld (Hrsg.): Information in Wissenschaft, Bildung und Wirtschaft, 29. Online-Tagung der DGI, 59. Jahrestagung der DGI, Proceedings, Frankfurt (Main): DGI, 2007, S. 121-136.

3. Katrin Weller, Indra Mainz, Ingo Paulsen, Dominic Mainz: Semantisches und vernetztes Wissensmanagement für Forschung und Wissenschaft. Erscheint in: WissKom 2007, Wissenschaftskommunikation der Zukunft, 4. Konferenz der Zentralbibliothek im Forschungszentrum Jülich, Proceedings, 2007.

Abstract

This thesis is about two topics: building a database of homologous invertebrate genes named INVHOGEN, and the creation of an Internet platform, ONTOVERSE, for collaborative ontology development and maintenance.

The first part of the thesis investigates the use of sequence similarity to group sequence entries into gene families. All gene families are explored by means of different annotation aspects such as species distribution, sequence distribution, and descriptions of the entries to characterize them with emphasis on Gene Ontology annotations. Traditional annotations written by scientists in natural language are partially suitable for machine processing. Ontological annotations of sequence entries promise to additionally represent knowledge computationally amenable to support the sequence based approach by semantic components with ontologies.

These results regarding ontological annotation quality, among other motivations, lead to the question how to bridge the two fields, database annotation and ontologies, for successful resource annotation of biological sequence data. For this purpose, an Internet-based application is created in the second part, that brings scientists (*domain experts*) together to offer them ways to communicate among each other or with *ontology designers*, which act as database curators in this special context. This collaborative approach should allow experts and engineers to improve database annotations by mutual understanding of the ontology's inner structure or even by the use of completely new designed ontologies, if special ontologies are desired for annotating sequence entries. Furthermore, existing ontologies might be extended by experts' knowledge to increase annotation qualitites.

While the widest use of bio-ontologies is for conceptual annotations, they are also used in a large range of other life science application scenarios which are manageable via the ONTOVERSE platform. The main focus in the second part of the thesis is on the architecture to manage scientific user communities and the integration of information extraction results into ontologies (*ontology population*).

Contents

Chapter 1

Introduction

As molecular biology (and several years later genomic projects) became very popular in life sciences, scientists began storing sequence information in dozens of large, publicly shared DNA sequence, protein and structure databases. One of the most significant data sources collaboratively maintained by the Swiss Institute of Bioinformatics and the European Bioinformatics Institute (EBI) is the SWISS-PROT protein information database [1]. Despite its origins as a simple sequence database, SWISS-PROT and its supplement TrEMBL have grown to include a wide spectrum of information about proteins in the form of annotations (e. g. three-dimensional structure, domains and sites, post-translational modifications, sequence conflicts, variants, etc.). Links from protein sequence entries to other large and disparate sources like organism specific databases, 2D-gel databases, 3D structure databases, genome annotation databases, enzyme and pathway databases and so on lead to another variety of information in SWISS-PROT.

From these sources the Gene Ontology (GO) developed at the GO Consortium [2] provides a framework for automatic functional annotation as an effective approach to associate individual sequences and related expression information with biological function. For example, the Gene Ontology Annotation (GOA) [3] project provides assignments of GO terms to SWISS-PROT and TrEMBL entries by a combination of electronic methods and manual annotation. Another research tool, Blast2GO (B2G) [4], enables GO based data mining on sequence data for which no GO annotation is available to support genomic research in non-model organisms. OntoBlast [5] and Goblet [6] assign GO terms to a new sequence based on its similarity to a sequence with a known GO assignment.

The similarity between GO terms can be used to compute a similarity between data entries that are annotated with these GO terms [7, 8].

Apart from annotating several database resources with e. g. GO, MGED (Microarray and Gene Expression Data)[1], or UMLS (Unified Medical Language System)[2] *ontologies* are used in a wide range of biomedical application scenarios [9]. For instance, they are used for providing visualization combining biological annotation with microarray expression data [10], metabolic pathways [11], medical image searching [12] and metasearches to biodiversity data [13].

Thesis outline

While Gene Ontology is generally accepted in life sciences, it also has some limitations regarding its internal structure. The GO hierarchy has highly varied depths along different branches — from two levels to more than 20 levels. Some of the variation is inherent in different functional families, while some may be an artifact of the uneven contribution by different groups participating in GO's development. This might be a source for mis-annotations in databases by biologists with little background to analyze and understand genes with the GO information.

Starting from this perspective, the main aim of this thesis is to develop concepts suitable to support scientists to collaboratively edit and maintain ontologies in life sciences. To demonstrate the practical use of the presented ideas, the Ontoverse platform [14] is developed.

The work presented in this thesis is outlined in the following. Background knowledge is introduced in chapter 2. This includes on the one hand the current state of ontologies in general and on the other hand ontologies in the life sciences with special regard to GO and the NCBI Taxonomy database. In addition, technical requirements specific to graphical

[1] http://www.mged.org

[2] http://www.nlm.nih.gov/research/umls/

user interface (GUI) application and Web application development are presented for the implementation of the applications described in the Chapters 3 and 4.

Chapter 3 shows the development of a database of homologous invertebrate genes named Invhogen [15]. This database integrates invertebrate protein sequences and annotations, taxonomic data, protein multiple sequence alignments (MSAs), and phylogenetic trees. With its graphical interface Jenfem, Invhogen allows one to rapidly and easily select sets of homologous genes and evaluate homology relationships between sequences. In the result section it is investigated amongst others if sequence similarity within gene families is correlated with semantic similarity of GO terms, i. e. where sequence similarity is very high, so does the chance that these proteins are homologues, in which case they are likely to be identically annotated.

In the first part of Chapter 4 the research project "Ontoverse – Collaborative knowledge management in the life sciences network", sponsored by the German Federal Ministry of Education and Research, is presented. Its central objective is the development of an Internet-based application for cooperative and interdisciplinary ontology building in terms of a so-called ontology wiki. The architecture of this kind of wiki is described in the second part of this chapter.

In the conclusion (Chapter 5) it is discussed how Ontoverse can contribute to fulfill the necessities for ontology engineering, annotating, and integrating for an upcoming Semantic Web.

Chapter 2

Background

The Semantic Web is a layer above the World Wide Web (WWW, Web) that adds meaning to hypertext links. In bioinformatics the Semantic Web addresses the dramatic increase of bioinformatics data available in Web-based systems and databases calls for novel processing methods. Furthermore, the high degree of complexity and heterogeneity of bioinformatics data and analysis requires semantic-based integration methods.

In this chapter at first XML (eXtensible Markup Language) and related technologies are presented. XML introduces structure to web documents, thus supporting syntactic interoperability. The structure of a document can be made machine-accessible through DTDs and XML Schema. With RDF and RDF Schema one can express statements between Web-based resources and data; it is a standard data model for machine-processable semantics. RDF Schema offers a number of modeling primitives for organizing RDF vocabularies in typed hierarchies. OWL, the current proposal for a web ontology language offers more modeling primitives, compared to RDF Schema, and has a clean, formal semantics. After this a survey of bio-ontologies is provided. These ontologies are concerned with biological and medical terminology and with ontologies for organizing other ontologies.

The rest of this chapter shifts the focus to desktop and web application development. The Cocoa framework and its Core Data infrastructure were used to implement a graphical user interface to access the INVHOGEN database in Chapter 3. The Ruby on Rails web application framework was chosen for the design of the ONTOVERSE platform in a RESTful style (Section 2.6).

2.1 Semantic Web

To make web pages understandable by machines, additional semantic information needs to be attached or embedded to the existing web data. Built upon the Resource Description Framework (RDF)[1], the Semantic Web is aimed at extending the current Web so that information can be given well-defined meaning using the description logic based ontology definition language OWL, and thus enabling better cooperation between computers and people. The Semantic Web can be viewed as a web of data that is similar to a globally accessible database.

The core of the Semantic Web are ontologies. They are used to capture the concepts and their relations in a domain for the purpose of information exchange and knowledge sharing. Over the past few years, several ontology definition languages have emerged, including RDF(S) and OWL. OWL is the newly released standard recommended by W3C[2].

The concept of the Semantic Web is to extend the current WWW such that context and meaning are given to information [16]. Instead of information being produced for machines, information will be produced for human consumption [17]. There are two main aspects of Semantic Web development: (1) ontologies for consistent terminology and (2) standards for interoperability (e. g. XML [18], RDF, HL7[3]).

Levels of Semantics

Semantics is the study of the meaning of signs, such as terms or words. Depending on the approaches, models, or methods used to add semantics to terms, different degrees of semantics can be achieved. This section identifies and describes four representations

[1]http://www.w3.org/RDF/
[2]http://www.w3.org
[3]http://www.hl7.org/

that can be used to model and organize concepts to semantically describe terms, that is, controlled vocabularies, taxonomies, thesauri, and ontologies. These four model representations are illustrated in Figure 2.1.

Controlled Vocabularies A controlled vocabulary is a list of terms (e. g. words, phrases, or notations) that have been enumerated explicitly. All terms in a controlled vocabulary should have an unambiguous, non-redundant definition. Controlled vocabularies are the simplest of all structured metadata methods and have been extensively used for classification.

Taxonomies They are subject-based classifications that arrange the terms of a controlled vocabulary into a hierarchy. The first users of taxonomies were biologists to classify organisms according to their natural relationships.

Thesauri A thesaurus is a networked collection of controlled vocabulary terms with conceptual relationships between them. A thesaurus is an extension of a taxonomy by allowing terms to be arranged in a hierarchy and also allowing other statements and relationships to be made about the terms.

Ontologies They are similar to taxonomies but use richer semantic relationships among terms and attributes, as well as strict rules about how to specify terms and relationships. In computer science, ontologies have emerged from the area of artificial intelligence. Ontologies have generally been associated with logical inferencing and recently have begun to be applied to the Semantic Web.

Technologies

The Semantic Web identifies a set of technologies, tools, and standards to provide a solid foundation for making the Web machine-readable. The Semantic Web infrastructure is based on several layers, each corresponding to a specific technology, and is commonly represented as 'layer cake'. A visual representation of the different parts of the Semantic Web architecture is displayed in Figure 2.2.

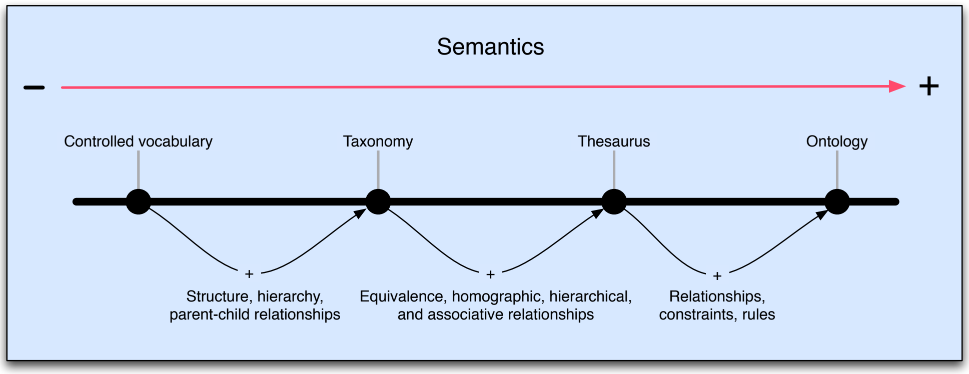

Semantics
–
+
Controlled vocabulary
Taxonomy
Thesaurus
Ontology
+
Structure, hierarchy, parent-child relationships
+
Equivalence, homographic, hierarchical, and associative relationships
+
Relationships, constraints, rules

Figure 2.1: Levels of semantics with increasing ways of expressing from left to right (*modified from* [19]).

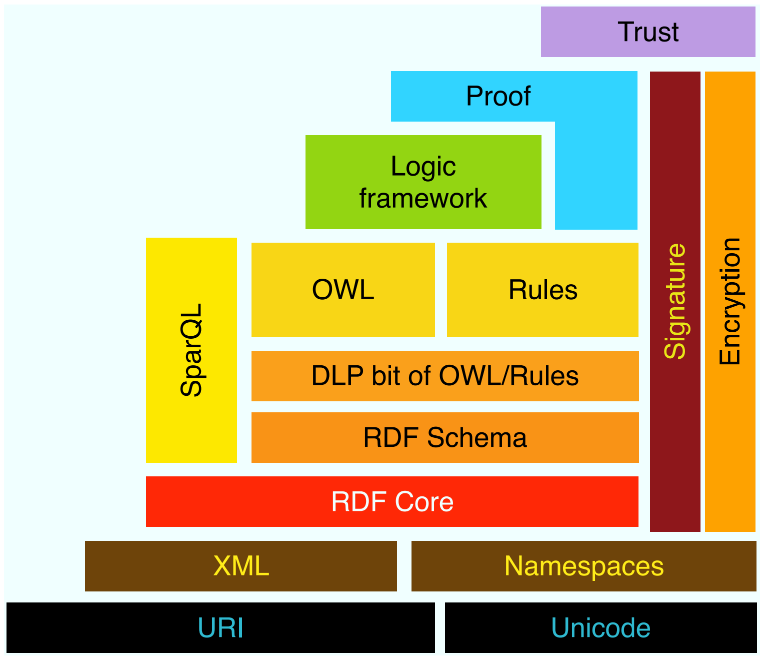


Figure 2.2: A layered approach to the Semantic Web. *Source: Tim Berners-Lee. Web for real people, 2005.* Available at `http://www.w3.org/2005/Talks/0511-keynote-tbl/`

The bottom layers in the layer cake, i. e. Unicode, URI (Uniform Resource Identifier), and XML (Schema), consist of existing web standards and provide a syntactical basis for Semantic Web languages. Unicode provides an elementary character-encoding scheme, which is used e. g. by XML (a standard syntax for structuring and describing data but not carrying any semantics). The URI standard provides a means to uniquely identify and address abstract or physical resources on the Web. All concepts used in the languages located higher in the layer cake can be specified using Unicode and are uniquely identified by URIs.

SPARQL is the emerging standard for querying and accessing RDF stores (Subsection 2.1.1). The Semantic Web Rule Language (SWRL) [20] allows data derivation, integration, and transformation.

The logic layer represents reasoning systems that infer new knowledge from ontologies and checks data consistency. The proof layer gives a proof of the logical reasoning conclusion by tracing the deduction of the interference engine. The trustfulness of Semantic Web information can be checked by the trust layer based on the signature and encryption layer. The proof and trust layers are currently under development, but most likely refer to the application and not to any specific language. For instance, the application could prove some statement by using deductive reasoning, and a statement could be trusted if it had been proven and digitally signed by some trusted third party.

2.1.1 RDF, RDFS, OWL

Resource Description Framework

At the top of XML the Resource Description Framework (RDF) is the first language developed especially for the Semantic Web. RDF was developed to add machine-readable metadata to existing data on the Web. RDF uses XML and it is at the base of the Semantic Web, so that all the other languages corresponding to the upper layers are built on top of it.

RDF is a general assertional model for representing explicit relationships between Web-based resources and data through RDF triples of *subject*, *predicate* and *object*.

The *subject* is the 'thing' being described, a *resource* identified by a URI in a common syntax regardless of the protocol is used to access the subject. The *predicate* is a property type of the resource, such as an attribute, a relationship, or a characteristic. The third component, *object*, is equivalent to the value of the resource property type for the specific subject. Each triple in RDF makes a distinct assertion, joining other triples will not change the meaning of the existing triples, regardless of the complexity of the model in which it is included. Figure 2.3 describes three statements using RDF triples.

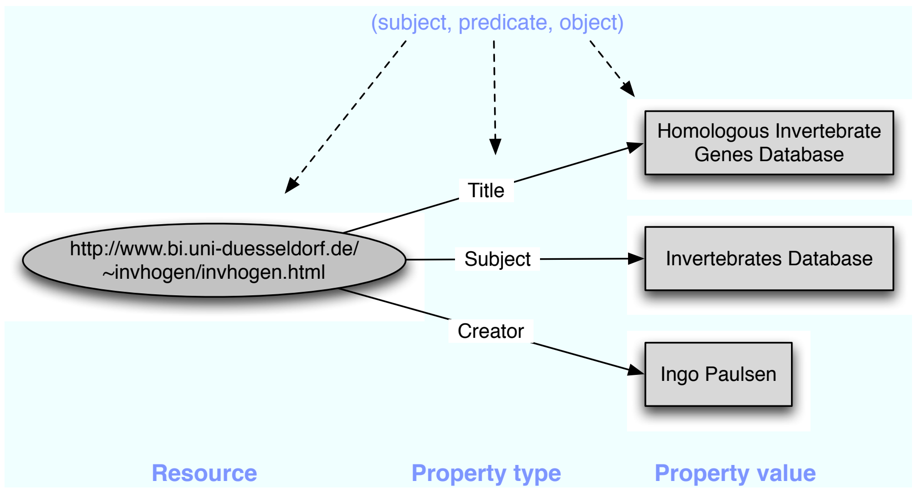

(subject, predicate, object)
Homologous Invertebrate Genes Database
Title
http://www.bi.uni-duesseldorf.de/~invhogen/invhogen.html
Subject
Invertebrates Database
Creator
Ingo Paulsen
Resource
Property type
Property value

Figure 2.3: Graphical representation of three RDF statements.

RDF Schema

RDF Schema (RDFS) is a domain-neutral lightweight schema language to define vocabularies for RDF. RDFS provides information about the interpretation of the statements given in an RDF data model. RDFS does not say anything about the syntactical appearance of the RDF description.

RDFS builds on the RDF foundation to provide additional descriptive features [21]. RDFS makes it possible to define a class, subclass, and with an instance being defined using `rdfs:Class`, `rdfs:subClassOf` and `rdf:type` respectively.

However, RDFS is not very expressive compared with other ontology languages, as it allows only the representation of concepts, concept taxonomies, and properties. OWL provides a richer set of vocabulary by further restricting on the set of triples that can be represented.

OWL

OWL (Web Ontology Language) is the standard web ontology language recently recommended by W3C. It is intended to be used by applications to represent terms and their interrelationships. OWL is used when information must be machine-processed and can be used to represent an ontology [22], as the RDF structure is unable to support a reasoner in using logical induction or deduction to infer new conclusions from statements.

OWL comes in three increasingly complex *species*: OWL Lite, OWL DL and OWL Full. OWL Lite offers a minimum number of features that are necessary to specify ontologies. It supports simple classifications, allowing only cardinalities of 0 or 1 and only minimal contraints. OWL DL as a superset of OWL Lite, supports more complex ontologies, but still has some restrictions to guarantee processing finishing in finite time using a DL reasoner. OWL Full, a superset of OWL DL, removes some restrictions from OWL DL, with no computational guarantees and the possibility of indefinite processing time.

Classes An OWL document can include an optional ontology header and any number of class, property, axiom, and individual descriptions. In an ontology defined by OWL, a named class is described by a class identifier via `rdf:ID`. An anonymous class can be described by value (`owl:hasValue`, `owl:allValuesFrom`, `owl:someValuesFrom`) or cardinality (`owl:maxCardinality`, `owl:minCardinality`, `owl:cardinality`) restriction on property (`owl:Restriction`); by exhaustive enumeration of all the individuals that form the instances of this class (`owl:oneOf`); or by logical operations on two or more other classes (`owl:intersectionOf`, `owl:unionOf`, `owl:complementOf`).

The three logical operators correspond to AND (conjunction), OR (disjunction) and NOT (negation) in logic define classes of all individuals by standard set operations of intersection, union, and complement, respectively. Three class axioms (`rdfs:subClassOf`, `owl:equivalentClass`, `owl:disjointWith`) can be used for defining necessary and sufficient conditions of a class.

Properties Two kinds of properties can be defined in an OWL ontology: object property (`owl:ObjectProperty`) which links individuals to individuals, and datatype property (`owl:DatatypeProperty`) which links individuals to data values. Similar to classes, `rdfs:subPropertyOf` is used to define that one property is a subproperty of another property. There are constructors to relate two properties (`owl:equivalentProperty` and `owl:inverseOf`), to impose cardinality restrictions on properties (`owl:FunctionalProperty` and `owl:InverseFunctionalProperty`), and to specify logical characteristics of properties (`owl:TransitiveProperty` and `owl:SymmetricProperty`). There are also constructors to relate individuals (`owl:sameAs`, `owl:sameIndividualAs`, `owl:differentFrom` and `owl:AllDifferent`).

The semantics of OWL is defined based on model theory in the way analogous to the semantics of description logic (DL). With the set of vocabulary (mostly as described above), one can define an ontology as a set of (restricted) RDF triples which can be represented as an RDF graph.

2.2 Ontologies

The word ontology has been borrowed from philosophy, where it means a systematic explanation of being. The knowledge engineering community has adopted ontology as a key enabling technology since the nineties. One of the first definitions of ontology given by Neches *et al.* [23], is as follows: "an *ontology* defines the basic terms and relations comprising the vocabulary of a topic area as well as the rules for combining terms and relations to define extensions to the vocabulary". According to the above definition, an ontology includes not only the terms that are explicitly defined in it, but also the knowledge that can be inferred from it. Gruber [24] defined an ontology as "an explicit specification of a conceptualization", which has become one of the most acceptable definitions to the ontology community. Guarino *et al.* [25] collected and analyzed seven definitions of ontology and provided their corresponding syntactic and semantic interpretations. They proposed to consider an ontology as "a logical theory which gives an explicit, partial account of a conceptualization", where conceptualization is basically an idea of the world that a person or a group of people can have.

Ontologies consist of definitional aspects such as high-level schemas and assertional aspects such as entities, attributes, interrelationships between entities, domain vocabulary and factual knowledge — all connected in a semantic manner [26]. Ontologies provide a common understanding of a particular domain. They allow the domain to be communicated between people, organizations, and application systems. Ontologies provide the specific tools to organize and provide a useful description of heterogeneous content.

In addition to the hierarchical relationship structure of typical taxonomies, ontologies enable cross-node horizontal relationships between entities, thus enabling easy modeling of real-world information requirements. Jasper and Uschold [27] identify three major uses of ontologies:

1. To assist in communication between human beings.

2. To achieve interoperability among software systems.

3. To improve the design and the quality of software systems.

An ontology is technically a model which looks very much like an ordinary object model in object-oriented programming. It consists of classes, inheritance, and properties [28]. In many situations, ontologies are thought of as knowledge representation.

Description logics are logical formalisms for knowledge representation [29]. They provide a formal linear syntax to express the description of top-level concepts in a problem domain; their relationships and the constraints on the concepts; and the relationships that are imposed by pragmatic considerations in the domain of interest [30, 31]. DL is divided into two parts: *Abox* (assertion component) and *Tbox* (terminological component). "Tbox vocabularies define concepts that have associated Abox facts. The combination of Tbox vocabularies and Abox facts represent a knowledge base" [32].

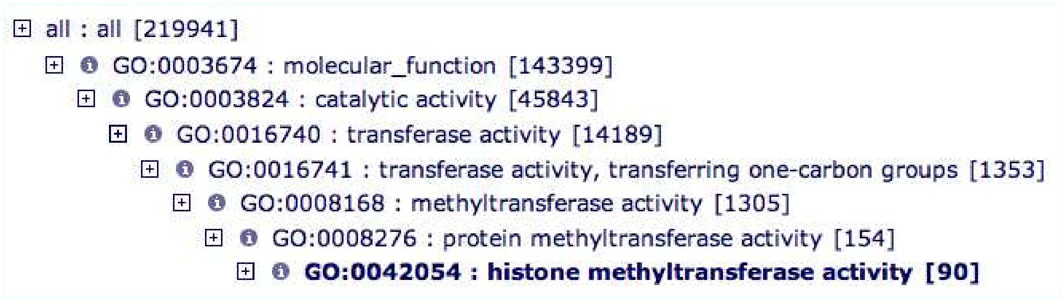
2.2.1 Bio-Ontologies

Ontologies have a very prominent role in bioinformatics since much of biology works by applying prior knowledge to an unknown entity. Within the last decade the research on ontologies has increased tremendously, and as a result more and more bio-ontologies become available. Therewith, to be of public value an ontology has to be widely disseminated and accepted by the field of knowledge that it models [33]. Moreover, in terms of inter-operability between databases and different scientific communities, the standard of ontologies becomes more and more important. The integration of information sources in the life sciences is one of the most challenging goals of bioinformatics.

Gene Ontology

GO is one of the most significant ontologies for bioinformatics and biology. The objective of GO is to supply a mechanism that guarantees consistent descriptions of gene products in different databases. GO is rapidly acquiring the status of a de facto standard in the field of gene and gene product annotations [34]. The GO effort includes the development of controlled vocabularies that describe gene products, establishing associations between the ontologies, the genes, and the gene products in the databases, and develop tools to create, maintain, and use ontologies. GO has over 20,000 terms and it consists of three distinct sub-ontologies that describe gene products in terms of their associated (1) molecular functions, (2) biological processes, and (3) cellular components [35]. Molecular function describes the tasks performed by individual gene products. Biological process describes broad biological goals that are accomplished by ordered assemblies of molecular functions. Cellular component encompasses subcellular structures, locations, and macromolecular complexes.

An example of the GO hierarchy for the term 'histone methyltransferase activity' is given in Figure 2.4. This shows the series of successively more restrictive concepts to which this term belongs.



```
⊞ all : all [219941]
  ⊞ GO:0003674 : molecular_function [143399]
    ⊞ GO:0003824 : catalytic activity [45843]
      ⊞ GO:0016740 : transferase activity [14189]
        ⊞ GO:0016741 : transferase activity, transferring one-carbon groups [1353]
          ⊞ GO:0008168 : methyltransferase activity [1305]
            ⊞ GO:0008276 : protein methyltransferase activity [154]
              ⊞ GO:0042054 : histone methyltransferase activity [90]
```

Figure 2.4: The GO hierarchy for histone methyltransferase activity. The brackets show the total number of GO terms in the category at that level.

GO did not originally make use of a formal ontological framework such as XML or RDF. To remedy this situation, the Gene Ontology Next Generation Project (GONG)[4] is developing a staged methodology to change the current representation of the GO into OWL. This allows one to take advantage of the richer formal expressiveness and the reasoning capabilities of the underlying formal logic [36].

Microarray Gene Expression Data

Another well-known life science ontology is the Microarray Gene Expression Data (MGED) ontology. MGED provides standard terms in the form of an ontology organized into classes with properties for the annotation of microarray experiments [37]. These terms provide an unambiguous description of how experiments were performed and enable structured queries over elements of the experiments. The comparison between different experiments is only feasible if there is standardization in the terminology to describe experimental setup, mathematical post-processing of raw measurements, genes, tissues, and samples. The adoption of common standards by the research community for describing data makes it possible to develop systems for the management, storage, transfer, mining, and sharing of microarray data [38].

[4]`http://www.gong.manchester.ac.uk/`

Open Biomedical Ontologies

The Open Biomedical Ontologies (OBO)[5] project forms an umbrella for a range of ontologies being designed for different biological and medical domains. The criteria for inclusion are that the ontology is open, uses either GO or OWL syntax, has definitions and unique identifiers, and complements other OBO ontologies. OBO contains various bio-ontologies ranging from anatomy to development, genomics, proteomics, and metabolomics, phenotype, taxonomic classification, and experimental conditions.

Biomedical Ontologies outside OBO Some other biomedical ontologies that were developed before OBO was established are:

- EcoCyc[6] is one of the oldest bio-ontologies and describes the metabolic and transduction pathways of *Escherichia coli* K12, its enzymes, and its transport proteins.
- OpenGalen[7] is an ontology used for medical information management.
- BioPAX[8] describes biological pathways and it is implemented in OWL.

2.2.2 Ontologies of Bioinformatics Ontologies

With the proliferation of biological ontologies and databases, the ontologies themselves need to be organized and classified.

TAMBIS

TAMBIS (Transparent Access To Multiple Bioinformatics Information Sources) [39] is a project that aims to help scientists by building a homogenizing layer on top of various biological information services. The TAMBIS Ontology (TaO) is a semantic network that covers a wide range of bioinformatics concepts. It contains description of the principal

[5]`http://obo.sourceforge.net/`
[6]`http://ecocyc.org/`
[7]`http://www.opengalen.org/`
[8]`http://www.biopax.org/`

concepts of molecular biology and bioinformatics: macromolecules; their motifs, their structure, function, cellular location, and the processes in which they act.

2.3 Cocoa

Cocoa is a complete set of classes and application programming interfaces (APIs) for building Mac OS X applications and tools [40]. Cocoa is divided into two main frameworks: Foundation framework and Application Kit.

The Foundation framework is a set of tools that represents fundamental data types, accessing system services, messaging, threading, and more. The Application Kit provides the functionality to build GUIs for Cocoa applications. It provides access to the standard interface components ranging from buttons, menus, and text fields to complete, prepackaged interfaces for print dialogs, file operation dialogs, and alert dialogs. It also provides higher-level functionality to implement multiple document applications, text handling, and graphics.

2.3.1 Design Patterns

Cocoa uses many design patterns that are descriptions of common object-oriented programming practices. Here is brief list of the design patterns which are used in the Jenfem application (Section 3.4).

Model-View-Controller The Model-View-Controller (MVC)[9] pattern is used extensively in the Application Kit to separate an application into logically distinct units: a model, which knows how to work with application data, the view, which is responsible for presenting the data to the user, and the controller, which handles interaction between the model and the view.

Delegation In this pattern, one object, the delegate, acts on behalf of another object. Delegation is used to alter the behavior of an object that takes a delegate. Delegation minimizes the need to subclass objects to extend their functionality.

[9] `http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html`

Target/action The target/action pattern decouples user-interface components with the objects (the targets) that implement their actions. In this pattern, an activated control sends an action message to its target.

Key-value coding This pattern provides an interface for accessing an object's properties indirectly by name.

Key-value observing A mechanism that allows objects to be notified of changes to specific properties of other objects.

Cocoa Bindings Provides a way to keep an attribute of a view synchronised with a property of a model object. Instead of connecting a control to instance variables and action methods, it connects a control directly to an object's value.

2.3.2 Objective-C

Cocoa's native language is Objective-C [41]. The Foundation and Application Kit frameworks are implemented in this language, and using Objective-C provides access to all features of the frameworks.

Objective-C is a highly dynamic, message-based object-oriented language. Consisting of a small number of additions to ANSI C, Objective-C is characterized by its deferral of many decisions until runtime, supporting its key features of dynamic dispatch, dynamic typing, and dynamic loading. These features support many of the design patterns Cocoa uses. Because it is an extension of C, existing C code and libraries can work with Cocoa-based applications.

2.3.3 Core Data

Core Data is a Cocoa framework and provides an infrastructure for managing object graphs, including support for persistent storage to a variety of file formats [42]. Object-graph management includes features such as undo and redo, validation, and ensuring the integrity of object relationships. Object persistence means that Core Data saves model objects to a persistent store and fetches them when required. The persistent store of a

Core Data application can range from XML files to SQL databases. Core Data is ideally suited for applications that act as front-ends for relational databases.

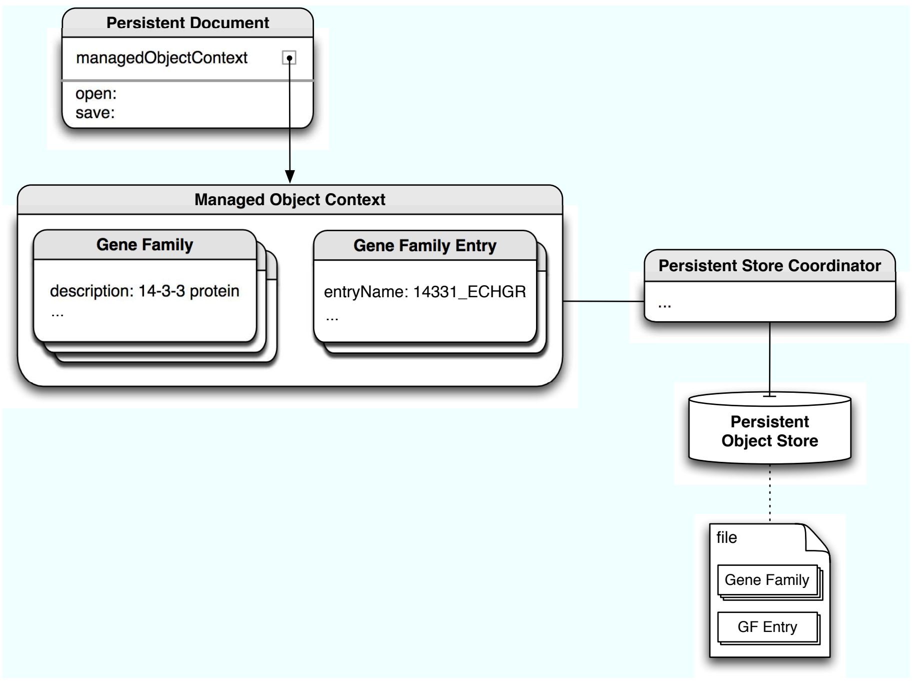


Figure 2.5: Document management using Core Data.

The central concept of Core Data is the Managed Object (MO). A MO is simply a model object that is managed by Core Data. One describes the MOs of a Core Data application using a schema called a Managed Object Model (MOM). A MOM contains descriptions of an application's managed objects (also referred to as *entities*). Each description specifies the *attributes* of an entity, its *relationships* with other entities, and *metadata* such as the names of the entity and the representing class.

In a running Core Data application, an object known as a Managed Object Context (MOC) is responsible for a graph of MOs. All MOs in the graph must be registered with a MOC. The context allows an application to add objects to the graph and remove them from it. It also tracks changes made to those objects, and thus can provide undo and redo support. When someone saves changes made to MOs, the MOC ensures that those objects are in a valid state. When a Core Data application wishes to retrieve data from its external data store, it sends a fetch *request* — an object that specifies a set of criteria — to a MOC. The context returns the objects from the store that match the request after automatically registering them.

A MOC also functions as a gateway to an underlying collection of Core Data objects called the *persistence stack*. The persistence stack mediates between the objects in an application and external data stores. It consists of two different types of objects, *persistent stores* and *persistent store coordinators*. Persistent stores are at the bottom of the stack. They map between data in an external store and corresponding objects in a MOC. They do not interact directly with MOCs, however. Above a persistence store in the stack is a persistent store coordinator, which presents a facade to one or more MOCs so that multiple persistence stores below it appear as a single aggregate store. Figure 2.5 shows the relationships between objects in the Core Data architecture.

2.4 Ruby

Ruby is a dynamically typed programming scripting language created by Yukihiro Matsumoto [43]. Ruby is completely object-oriented and allows to change classes and introducing new methods at runtime. It is a high-level programming language, less efficient but more flexible than compiled languages. It offers the following characteristics:

Interpreted Scripting languages are usually interpreted and not compiled, allowing quick turnaround development and making applications more flexible through runtime programming.

Reflection The possibility of easily investigating data and code during runtime, and runtime interrogation of objects instead of relying on their class definitions.

Metaprogramming Metaprogramming techniques allow code to be created, changed, and added during runtime. In Ruby, it is possible to change the behaviour of all objects during runtime and for example to add code to a single object (without changing its class).

Dynamic typing Scripting languages are usually weakly typed, without prior restrictions on how a piece of data can be used. Ruby has the so-called duck-typing mechanism in which object types are determined by their runtime capabilities instead of by their class definition.

2.5 Ruby on Rails

Ruby on Rails (Rails) is an open source web application development framework written in Ruby [44]. The goal of Rails is to develop web applications in an easy, straightforward manner, and with as few lines of code as necessary. By default, Rails makes a lot of assumptions and has a default configuration that works for most web applications. It is easy to override any defaults, but they are designed to keep initial application development simple.

2.5.1 MVC Architecture

Rails operates upon a subtly different variant of MVC architectural pattern called Model2. Model2 uses the same principles of MVC but tailors them for stateless web applications. This means that Rails applications are primarily split into three sections: models, views, and controllers. In Rails, these components have the following roles:

- *Models*: These are used to represent forms of data used by the application and contain the logic to manipulate and retrieve that data. In Rails, a model is represented as a class. Models are abstracted, idealized interfaces between controller code and data.

- *Views*: These are the templates and HTML code that users of the web application see. They turn data into a format that users can view. They can output data as HTML for browsers, XML, RSS, Atom, and other formats.

- *Controllers*: Controllers form the logic binding together models, data, and views. They process input and deliver data for output. Controllers call methods made available by models and deliver it to the views. They contain methods known as *actions* that, generally, represent each action relevant to that controller.

The basic relationship between these components is shown in Figure 2.6. The browser, on the client, sends a request for a page to the controller on the server. The controller retrieves the data it needs from the model in order to respond to the request. The controller renders the page and sends it to the view. The view sends the page back to the client for the browser to display.

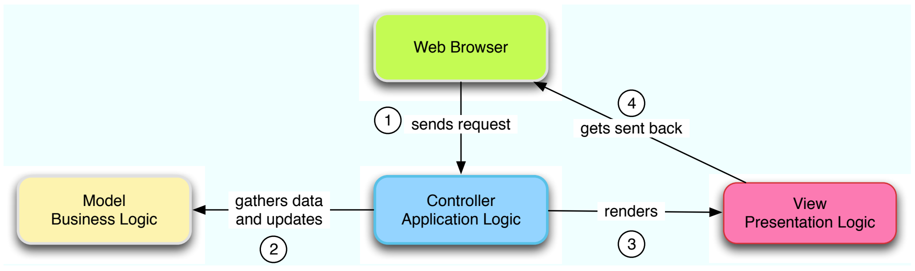


Figure 2.6: Processing a page request in a Ruby on Rails architecture.

2.5.2 Components of Rails

The Rails framework consists of several different libraries:

Rails The core library of the Ruby on Rails framework that ties the other libraries together.

ActionMailer A library that makes it easy to send e-mail from Rails applications.

ActionPack A library providing useful methods used in views and controllers to generate HTML and dynamic page elements, such as Ajax [45] and JavaScript, or manage data objects.

ActiveRecord An object-relational mapper (ORM) that ties database tables to classes.

ActiveSupport A library that collects support and utility classes used by various Rails features. For example, it supports methods for manipulating numbers, arrays, hashes, and times.

ActionWebService Provides methods to make it easy to offer functionality from Rails applications as a web service.

Figure 2.7 gives a schematic view of how Ruby and Rails fit these libraries together [46].

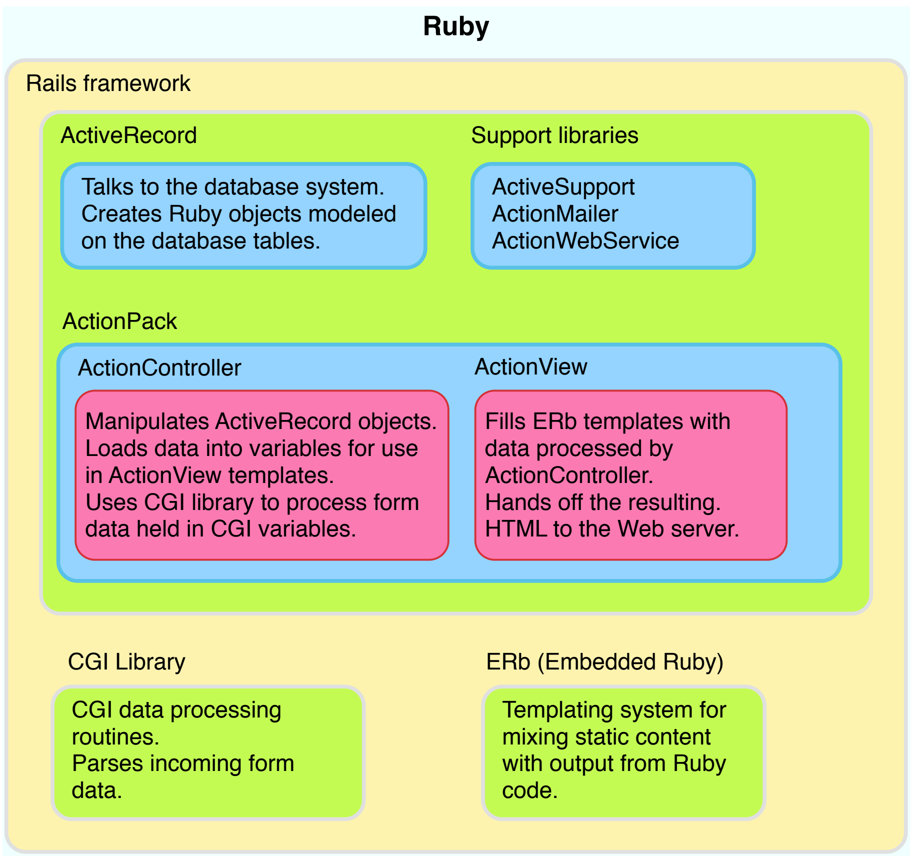

Ruby
Rails framework
ActiveRecord
Talks to the database system. Creates Ruby objects modeled on the database tables.
Support libraries
ActiveSupport
ActionMailer
ActionWebService
ActionPack
ActionController
Manipulates ActiveRecord objects. Loads data into variables for use in ActionView templates. Uses CGI library to process form data held in CGI variables.
ActionView
Fills ERb templates with data processed by ActionController. Hands off the resulting. HTML to the Web server.
CGI Library
CGI data processing routines. Parses incoming form data.
ERb (Embedded Ruby)
Templating system for mixing static content with output from Ruby code.

Figure 2.7: Schematic view of the Ruby on Rails framework.

2.6 RESTful Development

REST (REpresentational State Transfer) is a set of design criteria that was initially proposed by Roy Fielding [47]. It allows to build full-featured and extensible web services and applications on top of a small set of core, foundational operations. These operations are the four basic HTTP request methods (GET, POST, PUT, and DELETE), and the two auxiliaries (HEAD and OPTIONS). Web development has long ignored the full HTTP specification and only used GET and POST for requesting and sending data to and from dynamic web applications.

2.6.1 REST is a Conversation and Design

REST is about breaking down HTTP request to natural, human-language type structure with verbs and nouns. The verbs are the aforementioned request methods, while the nouns are URIs, unique identifiers for some resource[10] accessible via the Web. Every resource should have as few names as possible, and every name should be meaningful.

REST boils problems down to their bare essentials so that they can be addressed, analyzed, and represented properly. REST provides a framework for simple but extensible application design by mandating what actions an application can support against a resource:

- GET: retrieve a representation of a resource
- POST: create a new resource
- PUT: modify an existing resource
- DELETE: delete an existing resource

Many other common requests can be built on top of these verbs. Search is really the reading of resources that meet certain criteria.

[10]Something that can be stored on a computer and represented as a stream of bits.

2.6.2 REST and Rails

There are strong parallels between the REST verbs, the basic Rails controller actions, Create, Read, Update, Delete (CRUD), and the Add, Change, Inquire, Delete (ACID) operations of SQL. Table 2.1 shows how the verbs of ACID and HTTP correspond to each other.

CRUD:	create	read	update	delete
HTTP:	POST	GET	PUT	DELETE
Rails:	create	find	update	destroy
SQL:	INSERT	SELECT	UPDATE	DELETE

Table 2.1: CRUD, HTTP, Rails, SQL verbs.

When a resource is requested, the actual resource itself is not sent back to the user. Instead, a representation of that resource is sent back, often a web page describing the resource, or an image of it, or an XML document that structures the resource or the outcome of the action performed. This is represented in Figure 2.8.

With Rails, these various resource representations are built on top of controller actions, allowing requests for various forms of resources to share common processing logic. The implementation is abstracted from the services provided.

2.7 Resource-Oriented Architecture

The Resource-Oriented Architecture (ROA) is a way of turning a problem into a RESTful web service: an arrangement of URIs, HTTP, and XML that works like the rest of the Web. It has the four concepts resources, their names, their representations, and the links between them. Furthermore, it has the four properties addressability, statelessness, connectedness, and the uniform interface which are defined the following [48]:

Addressability An application is addressable if it exposes the interesting aspects of its data as resources. Since resources are exposed through URIs, an addressable application exposes a URI for every piece of information it might conceivably serve.

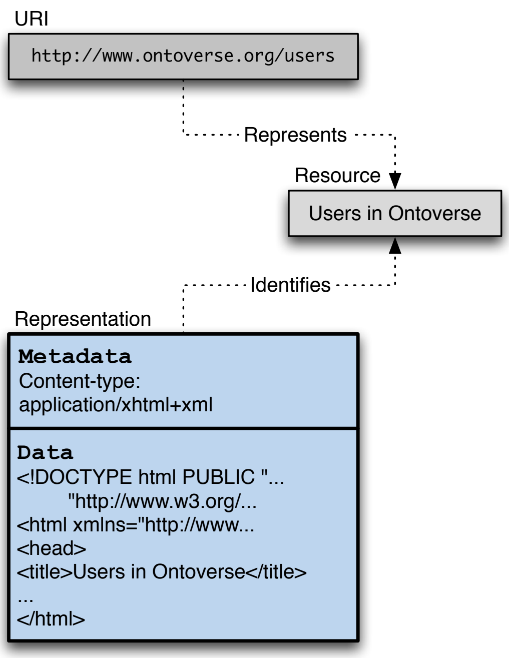


Figure 2.8: Requesting a resource. The relationships between identifier, resource, and representation.

Statelessness Means that every HTTP request happens in complete isolation. When the client makes an HTTP request, it includes all information necessary for the server to fulfill that request. The server never relies on information from previous request.

Connectedness A server can guide the client from one application state to another by sending links and forms in its representation. This also holds true for resource which should link to each other in their representations.

The Uniform Interface All interaction between clients and resources is mediated through a few basic HTTP methods. Any resource will expose some or all of these methods.

2.8 The Rails/ROA Design Procedure

2.8.1 RESTful Architecture of Rails

Routing

When an HTTP request comes in, Rails analyzes the requested URI and routes the request to the appropriate controller class. As shown in the following example, the file `config/routes.rb` tells Rails how to handle certain requests.

```
# routes.rb
ActionController::Routing::Routes.draw do |map|
  map.resources :projects do |project|
    project.resources :members
  end
end
```

That file declares the existence of two controller classes (ProjectsController and MembersController), and tells Rails how to route incoming requests to those classes. ProjectsController handles requests for the URI `/projects`, and for all URIs of the form `/projects/id`.

MembersController handles requests for the URI `/projects/project_id/members`, and all URIs of the form `/projects/project_id/members/id`.

Resources, Controllers, and Views

A Rails controller might expose a single *list* (or factory) resource, which responds to GET and/or POST requests, and a large number of *object* resources, which respond to GET, POST, and/or DELETE. The list resource often corresponds to a database table, and the object resources to the rows in the table.

Each controller is a Ruby class, so sending an HTTP request to a class means calling some particular method. Rails defines six standard methods per controller, as well as exposing two special view templates through HTTP GET. The seven HTTP requests made possible by the example above are:

1. GET `/projects`: A list of the projects. Rails calls the `ProjectsController#index` method.

2. GET `/projects/new`: The form for creating a new project. Rails renders the view in `app/view/projects/new.rhtml`.

3. POST `/projects`: Create a new project. Rails calls the `ProjectsController#create` method.

4. GET `/projects/id`: A project. Rails calls `ProjectsController#show`.

5. GET `/projects/id`;edit: The form for editing a project's state. Rails renders the view in `app/view/projects/edit.rhtml`.

6. PUT `/projects/id`: Change a project's state. Rails calls `ProjectsController#update`.

7. DELETE `/projects/id`: Delete a project. Rails calls `ProjectsController#delete`.

It is not necessary to expose all seven access points in every controller if not useful.

Chapter 3

INVHOGEN

This chapter describes the development of a database of homologous invertebrate genes named Invhogen from two protein sequence resources. In the method section the building steps, beginning from reducing inappropriate sequence entries and ended by meaningful namings of gene families, are performed. The interesting questions about the distribution of species and the annotation with GO terms in each gene family (GF) are investigated in the result part.

Afterwards the creating process of the graphical interface from Invhogen named Jenfem is pointed out. It allows one to rapidly and easily select homologous genes and evaluate homology relationships between sequences.

Eventually, on the basis of the results, it is discussed how annotation quality as one important part of knowledge resource can be improved by closer cooperation of scientists. As a consequence this discussion leads to Chapter 4, in which the Ontoverse approach is presented as a development and maintenance platform for multiple ontology projects.

3.1 Introduction

Genome projects [49] are generating an enormous amount of data in molecular and evolutionary biology. One goal of functional genomics is to determine the function of proteins predicted by these sequencing projects [50]. To overcome the problem of assigning protein functions to sequences one approach is to classify them into GFs on the basis of the presence of shared features or by clustering using some similarity measures under the assumption that proteins within the same GF possess similar or identical biochemical functions. To determine the function of new proteins one can infer its function or detect its functional regions by homology to other sequences. (If two proteins share a significant sequence similarity, then one typically concludes that they are probable to have similar function.) However, there are some cases where conserved structures within a protein group do not necessarily imply that these proteins perform the same function [51] owing to low-complexity sequences, multifunctional sequences and gene recruitment [52].

GFs are generated using sequence clustering. Sequence clustering allows the detection of all pair-wise sequence similarities within a given set of protein sequences. Proteins are then clustered into families based on their sharing of significant sequence similarity patterns. When sequence clustering is performed accurately, proteins within a family may be considered as sharing a common evolutionary history and possibly similar or identical functions [53].

Within a GF one has to distinguish between two types of homologies: genes are said to be orthologues in two different species if gene copies originate from a common ancestral gene after a speciation event. Paralogues are genes in a given species pair that diverged after duplication of an ancestral gene [54]. The distinction between paralogy and orthology is essential for molecular phylogeny since it is necessary to work with orthologous genes to infer species phylogeny from gene phylogeny.

To address the problem of detecting homologous genes, the INVertebrate HOmologous GENes (INVHOGEN) database was built. This database complements the three homologous databases HOVERGEN [55] devoted to vertebrates, HOBACGEN [56] devoted to prokaryotes and HOGENOM devoted to completely sequenced organisms. INVHOGEN contains the available invertebrate protein sequences from UniProt organized into families of homologous genes defined by sequence similarity. For many GFs INVHOGEN provides an MSA, a phylogenetic tree and taxonomic information about the sequences.

3.2 Methods

The second release of Invhogen has been built from the invertebrate entries in UniProt Release 5.5 [57] consisting of SWISS-PROT Release 47.5 and TrEMBL Release 30.5. The data consist of 284,763 protein entries, 11,702 of them from SWISS-PROT and 273,061 from TrEMBL. From both sequence files a total of 174,958 invertebrate protein entries were extracted. The SWISS-PROT/TrEMBL protein entries were used owing to their high level of annotation and integration with other databases, and of their minimal level of redundancy. By following the references in the database cross-reference (DR) field of SWISS-PROT/TrEMBL annotations, the corresponding nucleotide sequences from [58] were also integrated in the database structure.

For building the families, the BLASTP2 [59] program was applied to identify common regions between proteins, and to collect related proteins. A similarity search of all proteins against each other was performed by filtering low complexity regions with SEG [60], and using the BLOSUM62 amino acid similarity matrix [61] and an E-value threshold of 10^{-4}.

3.2.1 Gene Family Building

The results from BLASTP2 output are processed this way (Figure 3.2):

1. For each pair of sequences, high-scoring segment pairs (HSPs) that are not compatible within a global alignment are removed (Figure 3.1). The number of HSPs is reduced from 23,901,247 HSPs to 20,933,392 HSPs. 7,114,334 HSPs are originated from complete sequences and the rest belongs to fragments.

2. Two sequences in a pair are included in the same family if (right branch after HSP cutting step):

    - Both sequences are complete.
    - The remaining HSPs cover at least 80% of the proteins length.

- Their similarity is greater or equal to 50% (two amino-acids are considered similar if the BLOSUM62 similarity score is positive). This procedure reduces the risk of mis-assigning proteins with a complex evolutionary history involving gene fissions and fusions, and domain shuffling [51].

After this procedure only 24.7% (1,712,191 from 6,934,240) of the HSPs are prospects for building GFs.

3. Once families of complete protein sequences have been built, partial sequences or fragments (longer than 100 amino acids [62] or at least 50% of the length of the complete proteins) are included in the classification. A partial sequence matching with a complete protein is included in its family if (left branch after HSP cutting step):

   - The remaining HSPs cover at least 80% of the partial protein length.
   - Their similarity is greater or equal to 50%.

4. Short partial sequences (less than 100 amino acids and less than 50% of the length of the complete proteins) are not included in the classification.

5. Remaining 1,139,450 HSPs from complete sequences and 5,220,240 from fragments are combined for the clustering step into GFs, finally reduced due to redundancies to 6,124,427 HSPs.

6. Transitive links to build the families: If two pairs of sequences named A + B and B + C fulfill the conditions listed before, then A, B and C are integrated in the same family, this even if the pair A + C does not fulfill these conditions.

3.2.2 Naming of Gene Families

GFs are named using a written program that parsed the sequence description (DE) and similarity comment fields (SIMILARITY) of the SWISS-PROT/TrEMBL annotations. In the first step DE entries are clustered into subgroups of similar word orders. Each

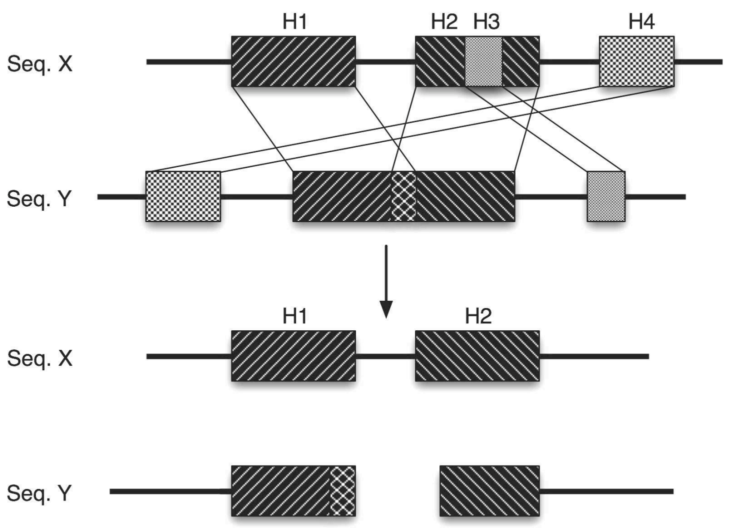


Figure 3.1: Removing incompatible HSPs. For each pair of sequences X and Y that hit each other using BLASTP2, HSPs that are not compatible with a global alignment are removed. In this example, hits H1 and H2 are compatible. However H3 and H4 are not compatible. Therefore, only H1 and H2 are considered for further computations on similarity measures. Because H1 and H2 are overlapping, the overlap is allocated to H1 and H2 is shortened accordingly. In a crossing-over situation between H1 and H2 for the sequences X and Y, H1 will be used if *length*(H1) > *length*(H2), otherwise, H2 is to take into account.

subgroup is named by assigning the most frequent position of every word and by joining these words together. A family description is created by combining all subgroup names considering only those with a large number of non-redundant entries in relation to the other subgroups. In the second step particular families are completed by available similarity comment lines for clarification reasons or if subgroup names are too different among themselves. Manual expertise is used to specify the name for a GF if both attempts failed to generate a meaningful name.

3.2.3 Multiple Sequence Alignments & Phylogenetic Trees

For each GF with at least four sequences, a MSA and a phylogenetic tree were built. Protein sequences were aligned with ClustalW 1.82 [63] with default parameters. Phylogenetic trees were reconstructed with IQPNNI 2.6 [64] by considering the so-called

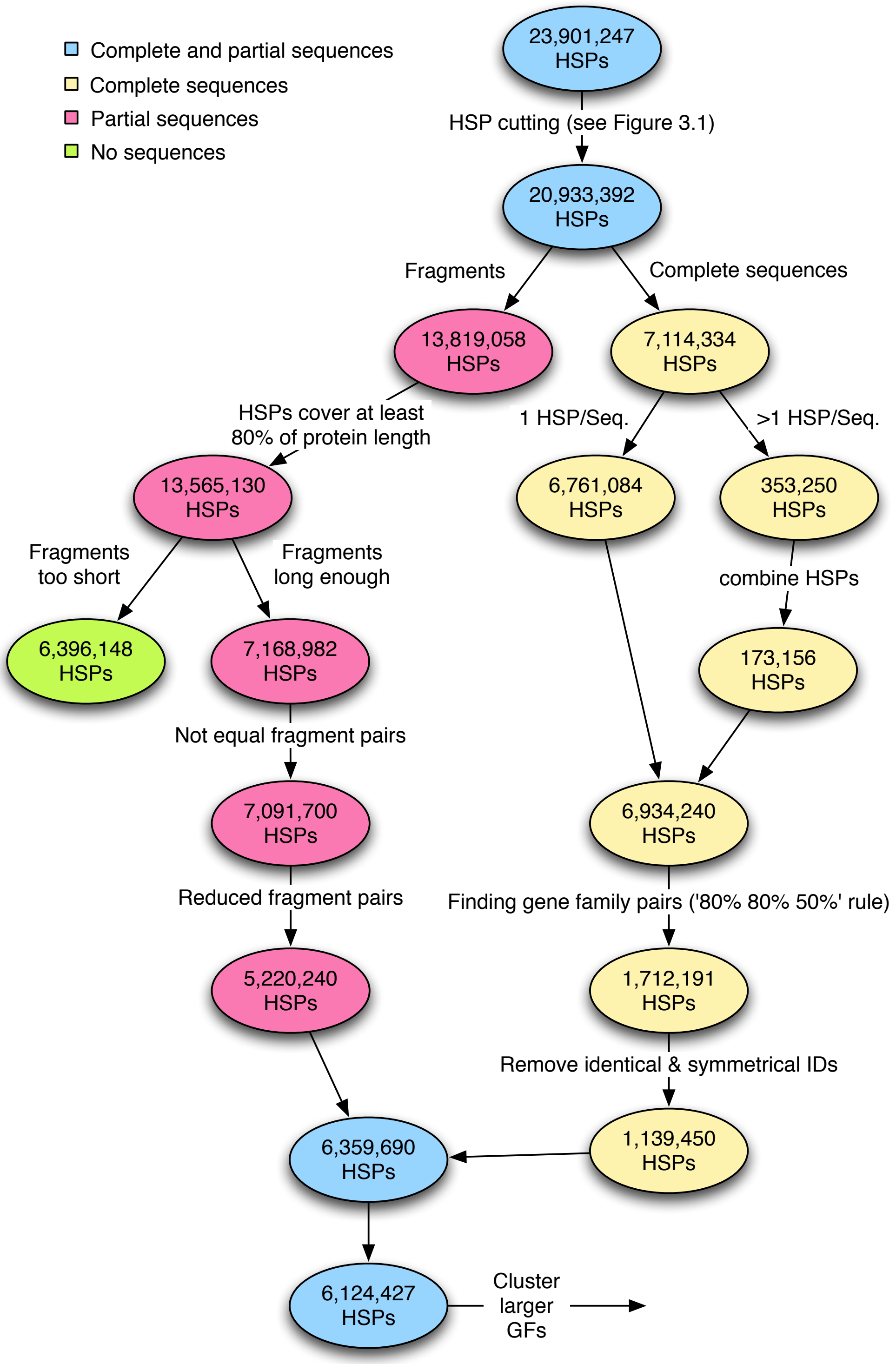


Figure 3.2: Reduction steps for HSPs of partial (left branch) and complete (right branch) protein sequences from BLASTP2 output. Details are given in Section 3.2.1.

stopping rule with at most 100 iterations. The stopping rule decides whether it is probable (with a 95% confidence level) that a continuation of the search will lead to no further improvement.

3.3 Results

3.3.1 Gene Family Distribution

The present version of Invhogen contains a total of 174,958 protein sequences (and 159,922 nucleic sequences) classified into 15,389 families. Among all the proteins included in this release 132,556 (75.8%) are classified into 15,389 families containing at least two sequences, and 42,402 (24.2%) partial proteins are not assigned to a family (so-called singletons). Table 3.1 shows the distribution of families in Invhogen grouped by family size in comparison with Hovergen. Table 3.2 displays the 10 largest families for both databases. These families consist of genes coding for proteins (or protein subunits) involved in protein translation, nucleotide biosynthesis, tissue development, and glycolysis. Cytochrome *c* oxidase polypeptide I, Cytochrome *b*, and NADH dehydrogenase subunit 1 are the only GFs that occur in both databases in the list of the top 10.

Family size	**No. of GFs INVHOGEN**		**No. of GFs HOVERGEN**	
2	8,567	(55.7%)	3,219	(24.5%)
3	2,257	(14.7%)	1,788	(13.6%)
4	1,210	(7.8%)	1,369	(10.5%)
5-9	2,093	(13.6%)	3,677	(28.0%)
10-19	693	(4.5%)	1,928	(14.7%)
20-49	358	(2.3%)	832	(6.3%)
50-99	116	(0.8%)	182	(1.4%)
$\geq$ 100	95	(0.6%)	149	(1.1%)
Total	15,389	(100%)	13,144	(100%)

Table 3.1: Distribution of GFs in Invhogen Release 2 and Hovergen Release 46.

GF Name INVHOGEN	Sequences		GF Name HOVERGEN
Cytochrome *c* oxidase polypeptide I	22,287	22,616	Cytochrome *b*
Cytochrome *c* oxidase polypeptide II	6,192	8,480	NADH dehydrogenase subunit 4
Cytochrome *b*	3,229	5,987	Family 1 of G-protein-coupled receptors
Elongation factor-1α	3,124	3,608	Class I histocompatibility antigen
NADH dehydrogenase subunit 1	1,586	2,990	ATP synthase subunit 6
NADH dehydrogenase subunit 5	1,568	2,291	ATP synthase subunit 8
WNT family	1,528	2,090	Cytochrome *c* oxidase polypeptide I
Serine peptidase	1,096	1,657	NADH dehydrogenase subunit 1
Homeobox protein	860	1,499	Zinc finger protein
Histone H3	836	1,314	NADH dehydrogenase subunit 6
Total	42,306	52,532	

Table 3.2: Ten largest GFs of Invhogen Release 2 and Hovergen Release 46.

3.3.2 Species Distribution

Table 3.3 presents the invertebrate and vertebrate species for which the greatest number of genes have been sequenced. Not surprisingly, species that are completely sequenced (e. g. *Drosophila melanogaster*, *Caenorhabditis elegans*) are the most frequent. They take up 44.5% of 132,556 protein sequences in Invhogen and 64.3% of the 214,379 sequences in Hovergen. Moreover, the distribution of all 22,053 species in Invhogen among all families is non-uniform. The first three species from Table 3.3 are overrepresented by at least 10,000 occurrences in number of sequences and appearance in families. However, 11,162 species only contribute a total of one sequence (data not shown).

The percentages of different classified species in the 12 main invertebrate groups and their representation in Invhogen are reported in Table 3.4. It is remarkable that the proportions of molluscs (13.52%), echinoderms (1.5%), and cnidarians (2.07%) in Invhogen are at least twice higher than the proportions reported in the literature (molluscs: 6.68%, echinoderms: 0.67%, cnidarians: 0.86%) [65]. The proportion of sequences in Invhogen for nematode sequences (22.56%) in comparison to all other species in Invhogen is a disproportionately high — owing to the completely sequenced genomes of *C. elegans* and *C. briggsae* — in comparison with the relative abundance of nematode species reported in the literature (1.43%) and in Invhogen (1.61%), respectively.

Species INVHOGEN	Sequences		Species HOVERGEN
Drosophila melanogaster *	17,348	56,932	*Homo sapiens* *
Caenorhabditis elegans *	16,604	46,693	*Mus musculus* *
Caenorhabditis briggsae *	10,704	9,066	*Rattus norvegicus* *
Anopheles gambiae PEST *	8,423	7,577	*Danio rerio* *
Schistosoma japonicum	2,143	5,392	*Xenopus laevis*
Drosophila simulans	998	3,258	*Gallus gallus*
Anopheles gambiae	894	3,038	*Bos taurus*
Bombyx mori *	689	2,790	*Sus scrofa*
Drosophila yakuba	608	1,720	*Macaca fascicularis*
Ixodes scapularis	538	1,325	*Oryctolagus cuniculus*
Total	58,949	137,791	

Table 3.3: The top 10 species in Invhogen Release 2 and Hovergen Release 46. (*) indicates the organisms where the complete genomic sequence is published (Genomes OnLine Database).

3.3.3 GO Term Annotations

In this section GO term annotations are examined in the face of annotations in protein sequences and on a GF distribution level. Furthermore, the annotation quantity is distinguished between appearance and how often an individual GO term occurs.

GO Term Distribution in Protein Sequences

Table 3.5 shows the distribution of all possible GO terms within all protein sequence entries in Invhogen. The number of GO terms are given in all odd columns indicated by #T and the number of sequence entries in even columns marked with Occ. (The number of 132,556 sequence entries results from the sum over all Occ. columns entries.) It is apparent that every fourth sequence entry is not annotated by any GO term (25.6%) at all and 44.3% of the sequences have references between one and five GO terms. From the remaining 39,806 sequence entries 39,196 sequences are annotated with at most nine GO terms, and only 0.46% (610 entries) have more than ten GO term references.

All protein sequences have a total of 489,717 GO terms with 3.69 annotations on average. They are dispersed into the three sub-ontologies as follows: 178,995 cellular components, 172,404 molecular function, 138,318 biological process.

Invertebrate groups	No. of Species/Fraction from Literature		in INVHOGEN		No. of Sequences/Fraction in INVHOGEN	
Arthropods	900,000	85.86%	16,681	77%	81,896	62.36%
Urochordates	3,000	0.29%	65	0.30%	910	0.69%
Echinoderms	7,000	0.67%	326	1.50%	2,718	2.07%
Poriferans	9,000	0.86%	112	0.52%	398	0.30%
Nematodes	15,000	1.43%	348	1.61%	29,630	22.56%
Platyhelminths	20,000	1.91%	369	1.70%	4,296	3.27%
Cnidarians	9,000	0.86%	448	2.07%	1,629	1.24%
Molluscs	70,000	6.68%	2,930	13.52%	8,088	6.16%
Annelids	15,000	1.43%	369	1.70%	1,041	0.79%
Hemichordates	100	0.01%	3	0.01%	74	0.06%
Cephalochordates	25	0%	8	0.04%	608	0.46%
Ctenophorans	150	0.01%	6	0.03%	31	0.02%
Total	1,048,275	100%	21,665	100%	131,319	100%

Table 3.4: Distribution of the main classified invertebrate groups in INVHOGEN Release 2 and from the literature [65].

Figure 3.3 represents the distribution of GO terms in all protein sequences. From the intention that each gene product should be annotated by classifying it three times, once within each sub-ontology [66], this distribution shows that 41.5% (55,081 entries) are annotated at most twice and 47.2% (62,544 entries) more than three times.

#T	Occ.	#T	Occ.	#T	Occ.	#T	Occ.	#T	Occ.
0	33,994	7	3,322	14	33	21	1	31	4
1	10,333	8	24,457	15	32	22	1	37	1
2	10,754	9	5,880	16	19	23	1	40	5
3	14,931	10	230	17	20	24	1	44	1
4	13,582	11	142	18	4	25	2	47	3
5	9,156	12	56	19	4	26	2	50	1
6	5,537	13	35	20	6	27	1	53	5

Table 3.5: Distribution of GO terms over the 132,556 sequence entries in INVHOGEN. #T represents the number of a GO term assignment within a sequence entry and Occ. shows how often each assignment counting appears in all INVHOGEN entries. For example, 10,333 sequence entries are annotated by just one GO term.

Table 3.6 gives an overview of the term annotations separated by the three sub-ontologies. As known from Table 3.5, e. g. 10,333 sequence entries have only one GO

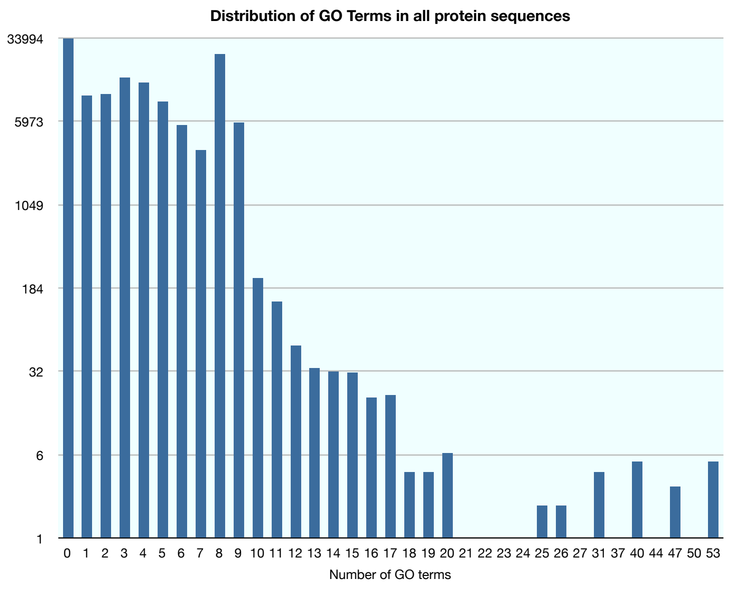


Figure 3.3: GO term distribution in all 132,556 protein sequences. The exact values for occurrences of GO terms are listed in Table 3.5.

annotation. In this case 6,161 sequence entries are annotated with a molecular function, 2,412 entries with cellular components and 1,760 of them with some kind of biological process. Table 3.6 also shows the composition for the very high number of 24,457 sequence entries with exactly eight occurrences. The first three largest values for all sub-ontologies are placed in row number eight on the left side.

Figure 3.4 illustrates the percentage of the GO sub-ontologies within the 34 classes with different numbers of annotation quantity. For the first seven numbers on the x-axis biological process terms are more frequently used than molecular function terms and cellular components (in this order). The highest appearances for the next two numbers of GO terms are for cellular components. Beginning with ten sequences a large number of GO term annotations are most commonly annotated with molecular functions.

GO Term Distribution in Gene Families

In addition to the distribution of GO terms on a sequence level Table 3.7 lists the distribution of GO terms per GF. From a total of 15,389 GFs 6,477 of them (42%) have

#T	P	F	C	#T	P	F	C
1	1,760	6,161	2,412	18	44	6	22
2	7,273	11,469	2,766	19	63	4	9
3	14,939	22,344	7,510	20	94	12	14
4	17,068	25,468	11,792	21	17	1	3
5	14,104	17,564	14,112	22	20	0	2
6	9,136	12,253	11,833	23	20	2	1
7	7,404	7,998	7,852	24	22	1	1
8	49,457	49,794	96,405	25	40	3	7
9	12,577	17,465	22,878	26	37	6	9
10	906	729	665	27	25	1	1
11	618	643	301	31	94	4	26
12	454	133	85	37	33	2	2
13	282	88	85	40	176	12	12
14	357	65	40	44	41	2	1
15	353	69	58	47	132	6	3
16	217	41	46	50	41	4	5
17	274	44	22	53	240	10	15

Table 3.6: Distribution of GO terms over the 132,556 sequence entries splitted into three sub-ontologies (P indicates biological process, F means molecular function, and C cellular component). The sum in each row for the three sub-ontologies is the same as the product of the values from the two columns identicated by #T and Occ. in the corresponding row in Table 3.5.

no annotation. Similar as for the sequence entries most GFs have between one and four term annotations (40%), 15.1% have five to nine annotations and 2.9% more than 10 with a maximum annotation number of 183 for GF `INV000838`. Figure 3.5 represents the distribution of GO terms in all GFs.

Most Frequent Annotated GO Terms

Table 3.8 shows the distribution of the 15 most frequent GO terms which occur at least once within a GF. Table 3.9 shows how often the 15 most frequent GO terms are annotated in all GFs totally. For instance, GO term nucleus (GO:0005634) appears in 1,549 GFs 6,877 times. Approximately every sixth GF (from 8,912 being annotated) carry this GO term 4,4 times on average. As the case of mitochondrion (GO:0005739), it occurs in only 347 GFs (in table row number 18, data not shown) but if so very often (119 times). Not surprisingly, the 15 most popular annotated GO terms are not far

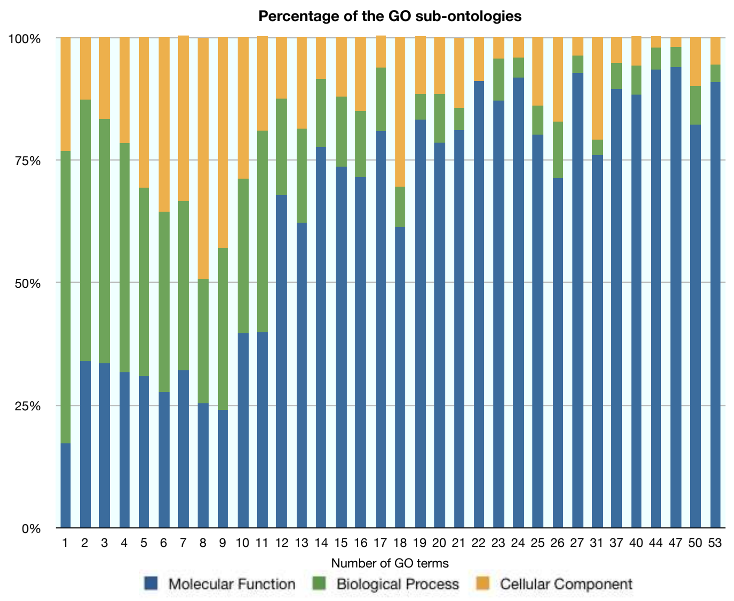


Figure 3.4: Percentage of the GO sub-ontology terms for all annoted protein sequence entries.

distant from the GO ontology root entry. This means, that the majority of all GFs are only annotated with more general GO terms, which does not distinguish them from other GFs on a GO annotation level.

#T	Occ.	#T	Occ.	#T	Occ.	#T	Occ.	#T	Occ.	#T	Occ.	#T	Occ.
0	6,477	7	353	14	24	21	3	29	4	41	1	53	1
1	1,633	8	246	15	27	22	10	30	1	42	2	54	1
2	1,478	9	154	16	16	23	3	31	2	43	1	55	1
3	1,783	10	84	17	14	24	5	33	2	44	1	59	2
4	1,268	11	65	18	14	25	5	34	1	47	1	66	1
5	950	12	49	19	21	26	2	38	1	49	2	117	1
6	622	13	43	20	9	27	2	40	1	52	1	183	1

Table 3.7: Distribution of all GO terms over the 15,389 GFs in Invhogen. #T represents the number of a GO term assignment within a GF and Occ. shows how often each assignment counting appears in all GFs. For example, 1,633 GFs are annotated by just one GO term.

.

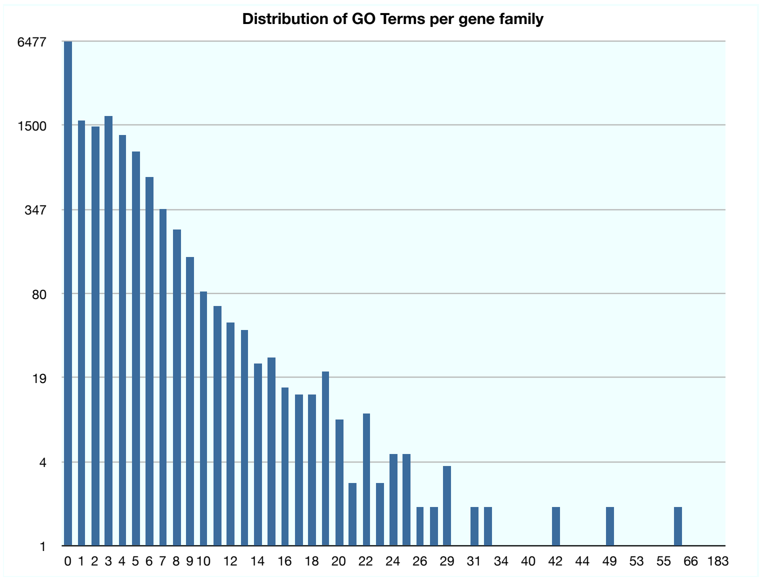


Figure 3.5: GO term distribution within all 15,389 GFs. The exact values for the number of GFs how often are assigned by GO terms are listed in Table 3.7.

GO Term	Occ.	Distance	Description	Sub-Ontology
0005634	1,549	4, 5, 6	nucleus	cellular component
0016021	1,046	5	integral to membrane	cellular component
0016020	896	3	membrane	cellular component
0006355	736	8	regulation of transcription	biological process
0005524	690	6	ATP binding	molecular function
0003676	619	3	nucleic acid binding	molecular function
0003677	595	4	DNA binding	molecular function
0008270	588	6	zinc ion binding	molecular function
0046872	498	4	metal ion binding	molecular function
0003700	494	3, 5	transcription factor activity	molecular function
0005515	492	3	protein binding	molecular function
0016740	475	3	transferase activity	molecular function
0006810	474	4, 5	transport	biological process
0016787	450	3	hydrolase activity	molecular function
0016491	402	3	oxidoreductase activity	molecular function

Table 3.8: Distribution of the number of occurrences of individual GO terms within all 15,389 GFs. 1,549 GFs are annotated with GO term GO:0005634 nucleus at least once.

GO Term	Occ.	Distance	Description	Sub-Ontology
0005739	41,373	5, 6	mitochondrion	cellular component
0016021	40,088	5	integral to membrane	cellular component
0016491	39,070	3	oxidoreductase activity	molecular function
0006118	37,211	5, 6	electron transport	biological process
0006810	32,898	4, 5	transport	biological process
0005746	30,342	6, 7, 8, 9	mitochondrial electron transport chain	cellular component
0004129	29,661	5, 6, 7	cytochrome-c oxidase activity	molecular function
0019866	28,022	4	inner membrane	cellular component
0016020	13,605	3	membrane	cellular component
0005634	6,877	4, 5, 6	nucleus	cellular component
0005507	6,208	6	copper ion binding	molecular function
0005525	5,510	6	GTP binding	molecular function
0006412	4,776	6, 7	protein biosynthesis	biological process
0046872	4,748	4	metal ion binding	molecular function
0005524	4,346	6	ATP binding	molecular function

Table 3.9: Distribution of the 15 most popular GO terms over all 15,389 GFs. 41,373 sequence entries are annotated with GO term GO:0005739 mitochondrion.

.

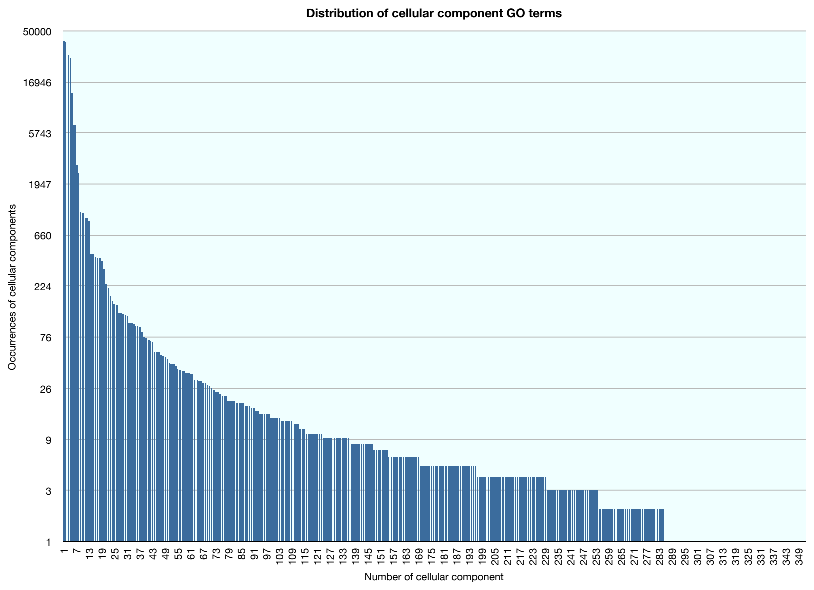


Figure 3.6: Distribution of cellular component GO terms over all 15,389 GFs. 352 cellular components from 1,695 cellular component terms in GO are assigned to at least one sequence entry.

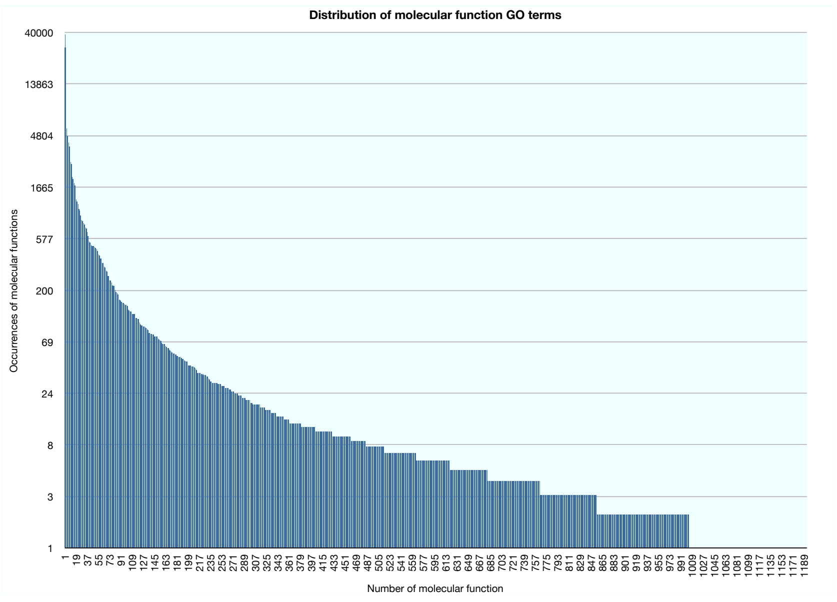


Figure 3.7: Distribution of molecular function GO terms over all 15,389 GFs. 1,193 molecular functions from 7,594 molecular function terms in GO are assigned to at least one sequence entry.

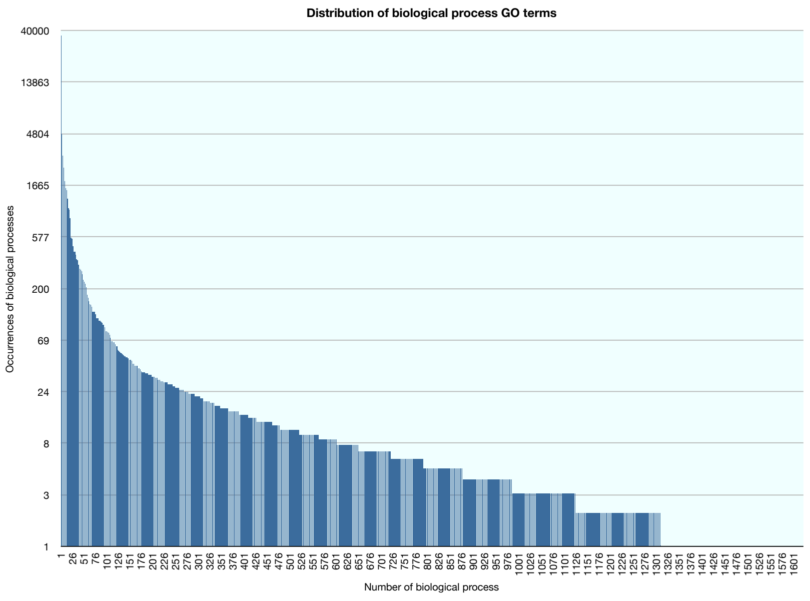


Figure 3.8: Distribution of biological process GO terms over all 15,389 GFs. 1,622 biological processes from 9,961 biological process terms in GO are assigned to at least one sequence entry.

3.4 Graphical Interface: Jenfem

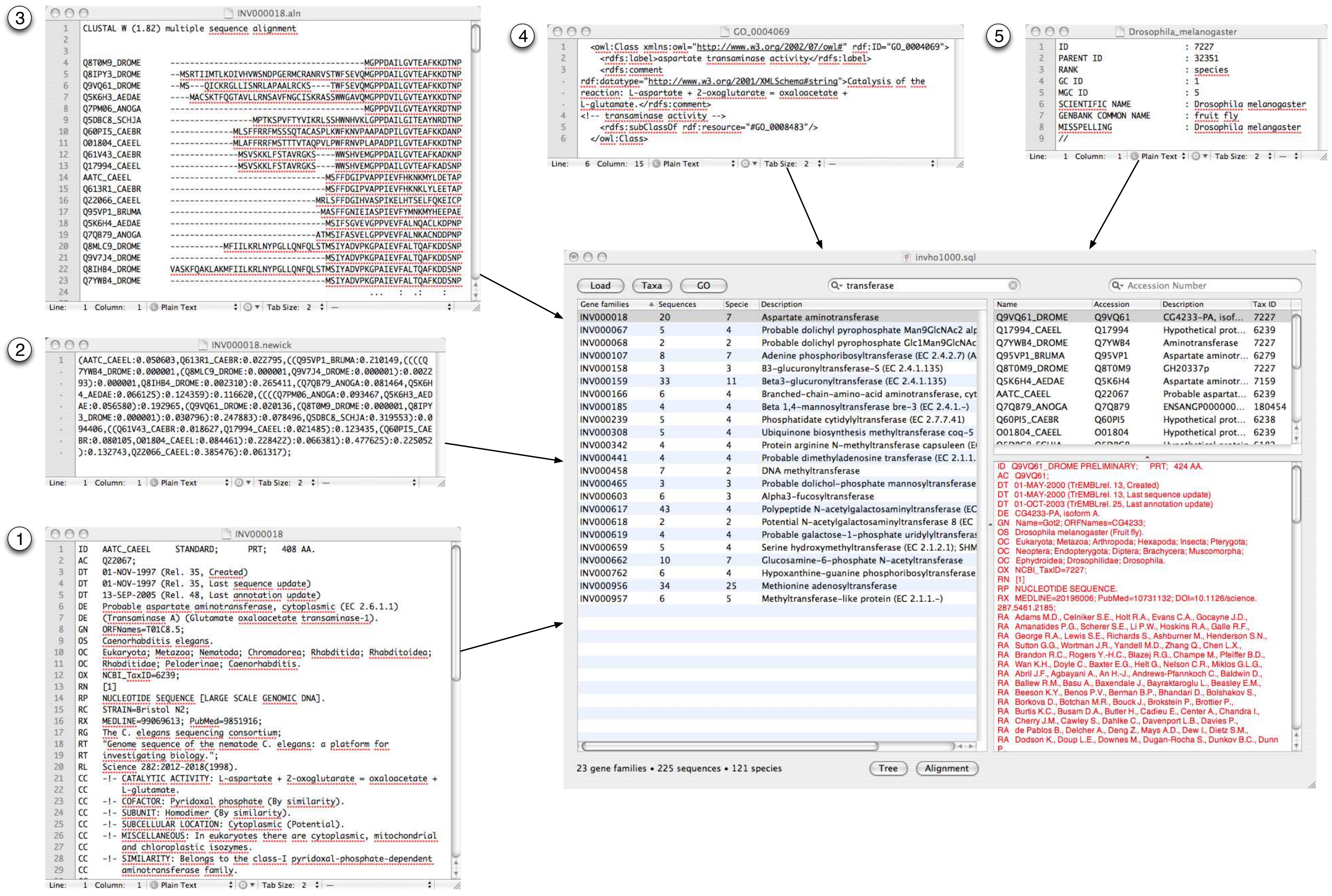
Jenfem is based on the Core Data architecture. It allows users to easily access and see the list of the GFs available in the database, the protein sequences of the genes in the families, the corresponding protein MSAs and the maximum likelihood based phylogenetic trees computed with these alignments. Furthermore, it offers a view which shows all annotated GO terms for a selected GF.

3.4.1 Data Integration

As shown in Figure 3.9 there exist several sources of legacy data to efficiently import them into the Jenfem application. The data consist of:

1. Protein sequence entries from SWISS-PROT/TrEMBL.
2. Phylogenetic trees in Newick format (if available for a GF).
3. Multiple sequence alignments (if available for a GF).
4. GO attributes.
5. Some entry fields from NCBI Taxonomy database.

The attributes taken from the GO and the taxonomy resources are listed in the data schema in Figure 3.10. All these 'flat' data are imported to create the MOs (Section 2.3.3) in a single pass, and then fixed up any relationships in a second pass.



```
CLUSTAL W (1.82) multiple sequence alignment


Q8T0M9_DROME      ------------------------------------MGPPDAILGVTEAFKKDTNP
Q8IPY3_DROME      --MSRTIIMTLKDIVHVWSNDPGERMCRANRVSTWFSEVQMGPPDAILGVTEAFKKDTNP
Q9VQ61_DROME      --MS---QICKRGLLISNRLAPAALRCKS----TWFSEVQMGPPDAILGVTEAFKKDTNP
Q5K6H3_AEDAE      ----MACSKTFQGTAVLLRNSAVFNGCISKRASSWWGAVQMGPPDVILGVTEAYKKDTNP
Q7PM06_ANOGA      ------------------------------------MGPPDVILGVTEAYKRDTNP
Q5DBC8_SCHJA      ----------------MPTKSPVFTYVIKRLSSHWNHVKLGPPDAILGITEAYNRDTNP
Q60PI5_CAEBR      ------------MLSFFRRFMSSSQTACASPLKWFKNVPAAPADPILGVTEAFKKDANP
O01804_CAEEL      ------------MLAFFRRFMSTTTVTAQPVLPWFRNVPLAPADPILGVTEAFKKDTNP
Q61V43_CAEBR      -------------MSVSKKLFSTAVRGKS----WWSHVEMGPPDAILGVTEAFKADKNP
Q17994_CAEEL      -------------MSVSKKLFSTAVRGKS----WWSHVEMGPPDAILGVTEAFKADSNP
AATC_CAEEL        ------------------------------MSFFDGIPVAPPIEVFHKNKMYLDETAP
Q613R1_CAEBR      ------------------------------MSFFDGIPVAPPIEVFHKNKLYLEETAP
Q22066_CAEEL      ----------------------------MRLSFFDGIHVASPIKELHTSELFQKEICP
Q95VP1_BRUMA      -----------------------------MASFFGNIEIASPIEVFYMNKMYHEEPAE
Q5K6H4_AEDAE      ------------------------------MSIFSGVEVGPPVEVFALNQACLKDPNP
Q7QB79_ANOGA      ----------------------------ATMSIFASVELGPPVEVFALNKACNDDPNP
Q8MLC9_DROME      ----------MFIILKRLNYPGLLQNFQLSTMSIYADVPKGPAIEVFALTQAFKDDSNP
Q9V7J4_DROME      ------------------------------MSIYADVPKGPAIEVFALTQAFKDDSNP
Q8IHB4_DROME      VASKFQAKLAKMFIILKRLNYPGLLQNFQLSTMSIYADVPKGPAIEVFALTQAFKDDSNP
Q7YWB4_DROME      ------------------------------MSIYADVPKGPAIEVFALTQAFKDDSNP
                                                     ...    :  .:    :
```

```
(AATC_CAEEL:0.050603,Q613R1_CAEBR:0.022795,((Q95VP1_BRUMA:0.210149,((((Q
7YWB4_DROME:0.000001,(Q8MLC9_DROME:0.000001,Q9V7J4_DROME:0.000001):0.0022
93):0.000001,Q8IHB4_DROME:0.002310):0.265411,(Q7QB79_ANOGA:0.081464,Q5K6H
4_AEDAE:0.066125):0.124359):0.116620,((((Q7PM06_ANOGA:0.093467,Q5K6H3_AED
AE:0.056580):0.192965,(Q9VQ61_DROME:0.020136,(Q8T0M9_DROME:0.000001,Q8IPY
3_DROME:0.000001):0.030796):0.247883):0.078496,Q5DBC8_SCHJA:0.319553):0.0
94406,((Q61V43_CAEBR:0.018627,Q17994_CAEEL:0.021485):0.123435,(Q60PI5_CAE
BR:0.080105,O01804_CAEEL:0.084461):0.228422):0.066381):0.477625):0.225052
):0.132743,Q22066_CAEEL:0.385476):0.061317);
```

```
ID   AATC_CAEEL      STANDARD;      PRT;   408 AA.
AC   Q22067;
DT   01-NOV-1997 (Rel. 35, Created)
DT   01-NOV-1997 (Rel. 35, Last sequence update)
DT   13-SEP-2005 (Rel. 48, Last annotation update)
DE   Probable aspartate aminotransferase, cytoplasmic (EC 2.6.1.1)
DE   (Transaminase A) (Glutamate oxaloacetate transaminase-1).
GN   ORFNames=T01C8.5;
OS   Caenorhabditis elegans.
OC   Eukaryota; Metazoa; Nematoda; Chromadorea; Rhabditida; Rhabditoidea;
OC   Rhabditidae; Peloderinae; Caenorhabditis.
OX   NCBI_TaxID=6239;
RN   [1]
RP   NUCLEOTIDE SEQUENCE [LARGE SCALE GENOMIC DNA].
RC   STRAIN=Bristol N2;
RX   MEDLINE=99069613; PubMed=9851916;
RG   The C. elegans sequencing consortium;
RT   "Genome sequence of the nematode C. elegans: a platform for
RT   investigating biology.";
RL   Science 282:2012-2018(1998).
CC   -!- CATALYTIC ACTIVITY: L-aspartate + 2-oxoglutarate = oxaloacetate +
CC       L-glutamate.
CC   -!- COFACTOR: Pyridoxal phosphate (By similarity).
CC   -!- SUBUNIT: Homodimer (By similarity).
CC   -!- SUBCELLULAR LOCATION: Cytoplasmic (Potential).
CC   -!- MISCELLANEOUS: In eukaryotes there are cytoplasmic, mitochondrial
CC       and chloroplastic isozymes.
CC   -!- SIMILARITY: Belongs to the class-I pyridoxal-phosphate-dependent
CC       aminotransferase family.
```

```
<owl:Class xmlns:owl="http://www.w3.org/2002/07/owl#" rdf:ID="GO_0004069">
  <rdfs:label>aspartate transaminase activity</rdfs:label>
  <rdfs:comment
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Catalysis of the
reaction: L-aspartate + 2-oxoglutarate = oxaloacetate +
L-glutamate.</rdfs:comment>
<!-- transaminase activity -->
  <rdfs:subClassOf rdf:resource="#GO_0008483"/>
</owl:Class>
```

```
ID                   : 7227
PARENT ID            : 32351
RANK                 : species
GC ID                : 1
MGC ID               : 5
SCIENTIFIC NAME      : Drosophila melanogaster
GENBANK COMMON NAME  : fruit fly
MISSPELLING          : Drosophila melangaster
//
```

Gene families	Sequences	Specie	Description
INV000018	20	7	Aspartate aminotransferase
INV000067	5	4	Probable dolichyl pyrophosphate Man9GlcNAc2 alp
INV000068	2	2	Probable dolichyl pyrophosphate Glc1Man9GlcNAc
INV000107	8	7	Adenine phosphoribosyltransferase (EC 2.4.2.7) (A
INV000158	3	3	B3-glucuronyltransferase-S (EC 2.4.1.135)
INV000159	33	11	Beta3-glucuronyltransferase (EC 2.4.1.135)
INV000166	6	4	Branched-chain-amino-acid aminotransferase, cyt
INV000185	4	4	Beta 1,4-mannosyltransferase bre-3 (EC 2.4.1.-)
INV000239	5	4	Phosphatidate cytidylyltransferase (EC 2.7.7.41)
INV000308	5	4	Ubiquinone biosynthesis methyltransferase coq-5
INV000342	4	4	Protein arginine N-methyltransferase capsuleen (E
INV000441	4	4	Probable dimethyladenosine transferase (EC 2.1.1.
INV000458	7	2	DNA methyltransferase
INV000465	3	3	Probable dolichol-phosphate mannosyltransferase
INV000603	6	3	Alpha3-fucosyltransferase
INV000617	43	4	Polypeptide N-acetylgalactosaminyltransferase (EC
INV000618	2	2	Potential N-acetylgalactosaminyltransferase 8 (EC
INV000619	4	4	Probable galactose-1-phosphate uridylyltransferas
INV000659	5	4	Serine hydroxymethyltransferase (EC 2.1.2.1); SHM
INV000662	10	7	Glucosamine-6-phosphate N-acetyltransferase
INV000762	6	4	Hypoxanthine-guanine phosphoribosyltransferase
INV000956	34	25	Methionine adenosyltransferase
INV000957	6	5	Methyltransferase-like protein (EC 2.1.1.-)

Name	Accession	Description	Tax ID
Q9VQ61_DROME	Q9VQ61	CG4233-PA, isof...	7227
Q17994_CAEEL	Q17994	Hypothetical prot...	6239
Q7YWB4_DROME	Q7YWB4	Aminotransferase	7227
Q95VP1_BRUMA	Q95VP1	Aspartate aminotr...	6279
Q8T0M9_DROME	Q8T0M9	GH20337p	7227
Q5K6H4_AEDAE	Q5K6H4	Aspartate aminotr...	7159
AATC_CAEEL	Q22067	Probable aspartat...	6239
Q7QB79_ANOGA	Q7QB79	ENSANGP000000...	180454
Q60PI5_CAEBR	Q60PI5	Hypothetical prot...	6238
O01804_CAEEL	O01804	Hypothetical prot...	6239

```
ID   Q9VQ61_DROME PRELIMINARY;    PRT;   424 AA.
AC   Q9VQ61;
DT   01-MAY-2000 (TrEMBLrel. 13, Created)
DT   01-MAY-2000 (TrEMBLrel. 13, Last sequence update)
DT   01-OCT-2003 (TrEMBLrel. 25, Last annotation update)
DE   CG4233-PA, isoform A.
GN   Name=Got2; ORFNames=CG4233;
OS   Drosophila melanogaster (Fruit fly).
OC   Eukaryota; Metazoa; Arthropoda; Hexapoda; Insecta; Pterygota;
OC   Neoptera; Endopterygota; Diptera; Brachycera; Muscomorpha;
OC   Ephydroidea; Drosophilidae; Drosophila.
OX   NCBI_TaxID=7227;
RN   [1]
RP   NUCLEOTIDE SEQUENCE.
RX   MEDLINE=20196006; PubMed=10731132; DOI=10.1126/science.
287.5461.2185;
RA   Adams M.D., Celniker S.E., Holt R.A., Evans C.A., Gocayne J.D.,
RA   Amanatides P.G., Scherer S.E., Li P.W., Hoskins R.A., Galle R.F.,
RA   George R.A., Lewis S.E., Richards S., Ashburner M., Henderson S.N.,
RA   Sutton G.G., Wortman J.R., Yandell M.D., Zhang Q., Chen L.X.,
RA   Brandon R.C., Rogers Y.-H.C., Blazej R.G., Champe M., Pfeiffer B.D.,
RA   Wan K.H., Doyle C., Baxter E.G., Helt G., Nelson C.R., Miklos G.L.G.,
RA   Abril J.F., Agbayani A., An H.-J., Andrews-Pfannkoch C., Baldwin D.,
RA   Ballew R.M., Basu A., Baxendale J., Bayraktaroglu L., Beasley E.M.,
RA   Beeson K.Y., Benos P.V., Berman B.P., Bhandari D., Bolshakov S.,
RA   Borkova D., Botchan M.R., Bouck J., Brokstein P., Brottier P.,
RA   Burtis K.C., Busam D.A., Butler H., Cadieu E., Center A., Chandra I.,
RA   Cherry J.M., Cawley S., Dahlke C., Davenport L.B., Davies P.,
RA   de Pablos B., Delcher A., Deng Z., Mays A.D., Dew I., Dietz S.M.,
RA   Dodson K., Doup L.E., Downes M., Dugan-Rocha S., Dunkov B.C., Dunn
P.,
```

23 gene families • 225 sequences • 121 species

Figure 3.9: Data integration into the JENFEM application. Five different data resources are used to store them in the Core Data schema and to gain access via the graphical interface. An example for a complete schema assignment is given in Table A.1.

3.4.2 Data Modeling

Data Modeling defines the data objects and their relationships, called a data schema. A schema defines entities that contain properties. Properties can be values (here *attributes*) or relationships to other entities. Entities (the containers defined in a data model) are like a class definition in object-oriented programming: They define the form from which any number of instances are created. It is different from a class in that it does not contain code. It is just a description of data.

Data Schema

In Jenfem there are five entities GeneFamily, GeneFamilyEntry, GeneOntologyTerm, Taxonomy, and Sequence. The central entity in this data schema is GeneFamilyEntry which has connections with all other entities, one-to-one or one-to-many. Each entity consists of a number of attributes and at least one relationship. Figure 3.10 shows the data schema with the five entities and the relationships between them. A single arrowhead represents a to-one relationship and a double arrowhead a to-many relationship. In both cases the arrow points to the destination entity. All relationships have been flagged as being inverse relationships, representing both relationships as a single line with two arrowheads. An inverse relationship does not just to make things more tidy, it is actually used by Core Data to maintain data integrity.

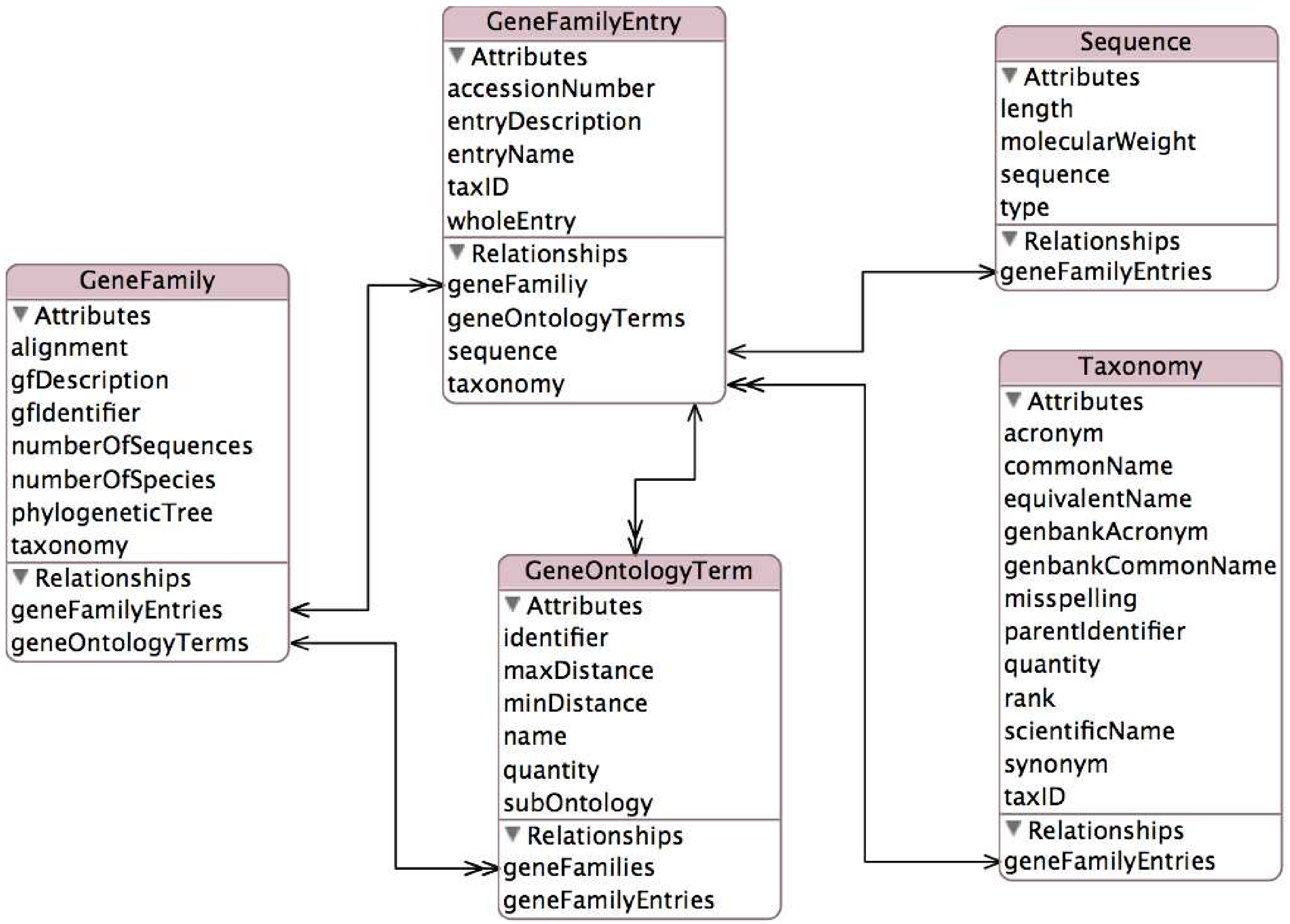


Figure 3.10: Data schema of Jenfem. Lines between entities describe relationships. The shapes of the arrowheads indicate the kind of relationship.

Name	**Type**	**Description**
alignment	String	Multiple Sequence Alignment
gfDescription	String	GF description
gfIdentifier	String	GF unique identifier
numberOfSequences	Int16	Number of sequences
numberOfSpecies	Int16	Number of species
phylogeneticTree	String	Phylogenetic tree
taxonomy	String	Path to all species or taxa within the GF

Table 3.10: GeneFamily table.

Data Tables

In the following each entity from the data schema is described in more detail. All tables consist of three columns and represent each attribute's name, its Core Data type and a description of the attribute. Three Core Data types are used, namely `String`, `Int16`, and `Int32`, where the last two are only differing in value range.

GeneFamily Table The GeneFamily table (Table 3.10) represents some aspects which characterize each GF: The number of sequences and species, a protein MSA, and a phylogenetic tree. The last two data representations are not specified if the number of sequences in a GF was less than four or more than 400. Furthermore, this table has a unique identifier and a description for each GF. A taxonomy string is used to allow not only to search for species but also for higher taxa (*query extension*).

An instance of a GF has two or more sequence entries and zero, one, or more GO term entries (represented by the relationships in Figure 3.10).

GeneFamilyEntry table The GeneFamilyEntry table (Table 3.11) contains attributes regarding a sequence entry which belongs to a GF. This table collects some important sequence entry fields from SWISS-PROT/TrEMBL like the accession number, the NCBI Taxonomy database ID [67, 68], entry name and description, and for completeness the original sequence entry as a string value.

A GF entry is connected to all other entities in a one-to-one relationship to Se-

Name	**Type**	**Description**
accessionNumber	String	Accession number of a sequence entry
entryDescription	String	Description of a sequence entry
entryName	String	The entry name
taxID	Int32	NCBI taxon identifier to which this gene product belongs
wholeEntry	String	Complete sequence entry from SWISS-PROT/TrEMBL

Table 3.11: GeneFamilyEntry table.

Name	**Type**	**Description**
identifier	String	The GO identifier
name	String	GO term name
subOntology	String	Indicates one of the three sub-ontologies (C, F or P)
quantity	Int16	Number of occurrences within at least one GF
minDistance	Int16	Minimum distance to GO's root entry
maxDistance	Int16	Maximum distance to GO's root entry

Table 3.12: GeneOntologyTerm table.

quence, in a many-to-one relationship to Taxonomy and GeneFamily, and in a zero-to-many relationship to GeneOntologyTerm.

GeneOntologyTerm Table The GeneOntologyTerm table (Table 3.12) represents some GO annotations for a given GF to show which GO terms all GF members have in common. Each GO term here is specified by a name, an identifier, and a sub-ontology (cellular component, molecular function or biological process). Additionally, the total number of GO term occurrences is given, as well as the minimum and maximum distances to GO's root entry. (Each GO sub-ontology is structured as a directed acyclic graph (DAG), wherein any term may have more than one parent as well as zero, one, or more children.)

A single GO term can be related to GF entries or to a collection of GF entries grouped into GFs.

Name	Type	Description
scientificName	String	Scientific name of a taxon
commonName	String	Common name
acronym	String	Acronym
equivalentName	String	Equivalent name
genbankAcronym	String	GenBank acronym
genbankCommonName	String	GenBank common name
misspelling	String	Misspelling(s) of a taxon
synonym	String	Synonym(s) of a taxon
rank	String	Rank of a taxon
quantity	Int16	Number of occurrences within all GFs
taxID	Int32	NCBI taxon identifier to which this GF entry belongs
parentIdentifier	String	The parent identifier

Table 3.13: NCBITaxonomy table.

NCBITaxonomy Table The NCBITaxonomy table (Table 3.13) stores the most common attributes of each taxon from the NCBI Taxonomy database. Four of them are mandatory (NCBI taxonomy identifier, parent identifier, rank, and scientific name). All the other fields are filled with values if required, e. g. an organism can have several different names (synonyms). The quantity attribute contains the number of occurrences for each taxon within one or a collection of GFs.

A taxonomy entry can be linked to more than one GF entry within a given family. In this case the total number of species is less than the number of sequences in a GF.

Sequence Table The Sequence table (Table 3.14) separately stores the protein sequences from all GF entries. Each sequence entry is composed of its sequence and sequence type, its length in amino acids, and the molecular weight in Dalton.

A sequence belongs to one GF entry and each GF entry has exactly one sequence.

Jenfem

MVC Aspects Jenfem represents the view from the MVC design pattern (whereas Core Data is responsible for the model part). In some respects every entity acts as a

Name	Type	Description
sequence	String	Sequence
type	String	Sequence type
length	Int32	Length of the sequence
molecularWeight	Int32	Molecular weight of the sequence

Table 3.14: SEQUENCE table.

data resource for a controller which on the other end is responsible for the correct display of the model in the view via Cocoa bindings.

Main Window The main window is organized in a master-detail interface. In the master interface portion, a table view is used to display the collection of GFs. In the detail interface portion, two views are used to display the GF entry's attributes of the selected GF. Figure 3.11 shows on the left side the table view of 17 GFs. In this example GF `INV000171` is selected for further inspection of its entries in the detailed views on the right side. The table view on the top right displays all 32 entries and on the bottom the sequence entry of the selected entry `BGBP2_DROME`.

Both table views in this window are connected with search fields to filter a collection of objects. The search field above the master portion can be used to filter GFs with a given description or to search for GFs with a certain taxon name. The other search field filters GF entries depending on accession numbers or descriptions.

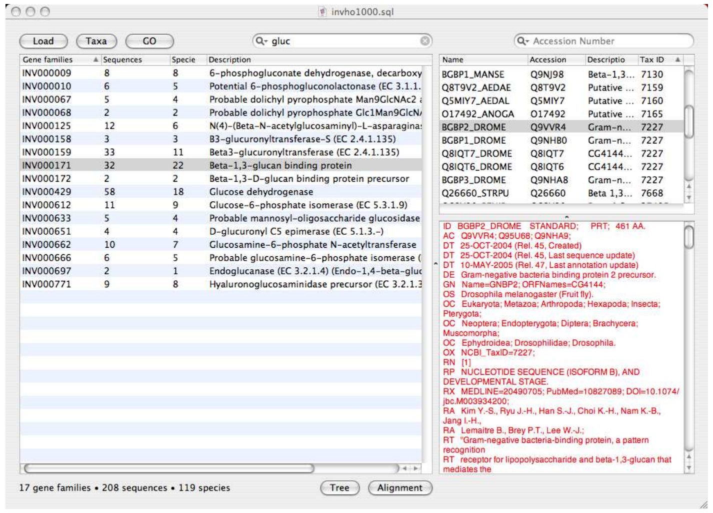
The buttons at the bottom of the main window are responsible for showing the alignment and the phylogenetic tree (if available) for a selected GF. The buttons at the top connect the main window with a taxonomy window of all taxa in INVHOGEN and a window which displays all GO terms of a GF.

Multiple Sequence Alignment Window This window displays the alignment of a given GF.

Phylogenetic Tree Window In this window the phylogenetic tree of a GF is displayed as a rectangular cladogram. At the bottom of this window the Newick tree format is also shown.

GO Terms Window The main portion of this window is a table view representing some details of every existing GO term in a GF (here `INV000171`). It shows that this GF is annotated with 12 different GO terms (one cellular component term, four molecular function terms, and seven biological process terms). All 32 sequences are annotated with a total of 97 GO terms with the highest quantity of 22 occurrences for two GO terms (GO:0004553, GO:0005975). The minimum distance of the GO root term is two for GO:0005576 term and the maximum distance is nine for GO:0042830 term.

A search field at the top of the window enables filtering GO terms depending on their descriptions.

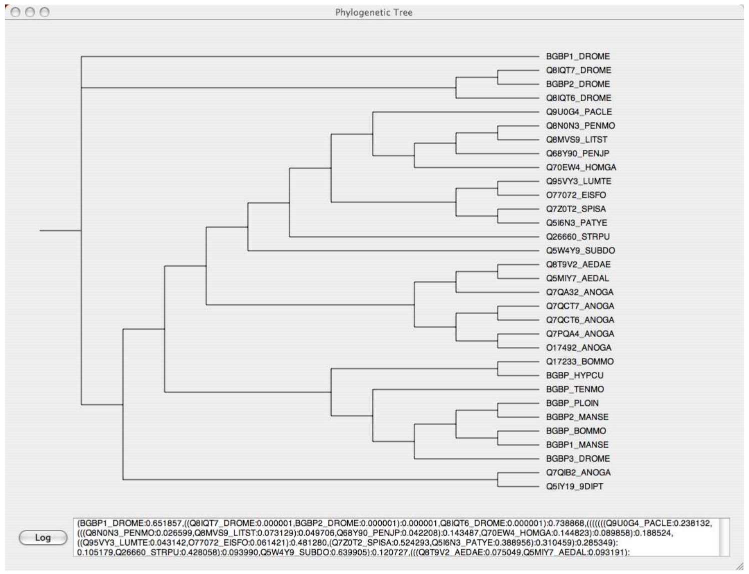


Gene families	Sequences	Specie	Description
INV000009	8	8	6-phosphogluconate dehydrogenase, decarboxy
INV000010	6	5	Potential 6-phosphogluconolactonase (EC 3.1.1.
INV000067	5	4	Probable dolichyl pyrophosphate Man9GlcNAc2 a
INV000068	2	2	Probable dolichyl pyrophosphate Glc1Man9GlcN/
INV000125	12	6	N(4)-(Beta-N-acetylglucosaminyl)-L-asparagina:
INV000158	3	3	B3-glucuronyltransferase-S (EC 2.4.1.135)
INV000159	33	11	Beta3-glucuronyltransferase (EC 2.4.1.135)
INV000171	32	22	Beta-1,3-glucan binding protein
INV000172	2	2	Beta-1,3-D-glucan binding protein precursor
INV000429	58	18	Glucose dehydrogenase
INV000612	11	9	Glucose-6-phosphate isomerase (EC 5.3.1.9)
INV000633	5	4	Probable mannosyl-oligosaccharide glucosidase
INV000651	4	4	D-glucuronyl C5 epimerase (EC 5.1.3.-)
INV000662	10	7	Glucosamine-6-phosphate N-acetyltransferase
INV000666	6	5	Probable glucosamine-6-phosphate isomerase (
INV000697	2	1	Endoglucanase (EC 3.2.1.4) (Endo-1,4-beta-gluc
INV000771	9	8	Hyaluronoglucosaminidase precursor (EC 3.2.1.3

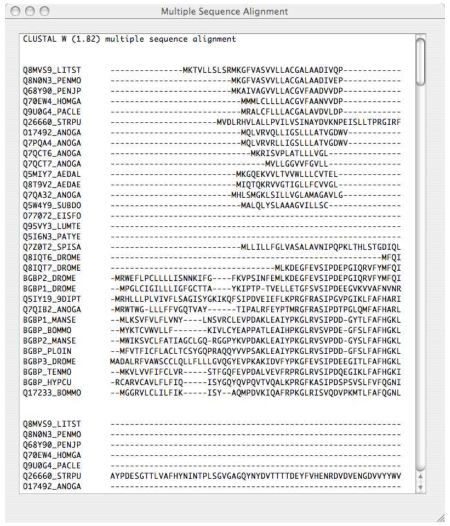
17 gene families • 208 sequences • 119 species

Name	Accession	Descriptio	Tax ID
BGBP1_MANSE	Q9NJ98	Beta-1,3...	7130
Q8T9V2_AEDAE	Q8T9V2	Putative ...	7159
Q5MIY7_AEDAL	Q5MIY7	Putative ...	7160
O17492_ANOGA	O17492	Putative ...	7165
BGBP2_DROME	Q9VVR4	Gram-n...	7227
BGBP1_DROME	Q9NHB0	Gram-n...	7227
Q8IQT7_DROME	Q8IQT7	CG4144...	7227
Q8IQT6_DROME	Q8IQT6	CG4144...	7227
BGBP3_DROME	Q9NHA8	Gram-n...	7227
Q26660_STRPU	Q26660	Beta 1,3...	7668

```
ID   BGBP2_DROME   STANDARD;    PRT;  461 AA.
AC   Q9VVR4; Q95U68; Q9NHA9;
DT   25-OCT-2004 (Rel. 45, Created)
DT   25-OCT-2004 (Rel. 45, Last sequence update)
DT   10-MAY-2005 (Rel. 47, Last annotation update)
DE   Gram-negative bacteria binding protein 2 precursor.
GN   Name=GNBP2; ORFNames=CG4144;
OS   Drosophila melanogaster (Fruit fly).
OC   Eukaryota; Metazoa; Arthropoda; Hexapoda; Insecta;
Pterygota;
OC   Neoptera; Endopterygota; Diptera; Brachycera;
Muscomorpha;
OC   Ephydroidea; Drosophilidae; Drosophila.
OX   NCBI_TaxID=7227;
RN   [1]
RP   NUCLEOTIDE SEQUENCE (ISOFORM B), AND
DEVELOPMENTAL STAGE.
RX   MEDLINE=20490705; PubMed=10827089; DOI=10.1074/
jbc.M003934200;
RA   Kim Y.-S., Ryu J.-H., Han S.-J., Choi K.-H., Nam K.-B.,
Jang I.-H.,
RA   Lemaitre B., Brey P.T., Lee W.-J.;
RT   "Gram-negative bacteria-binding protein, a pattern
recognition
RT   receptor for lipopolysaccharide and beta-1,3-glucan that
mediates the
```



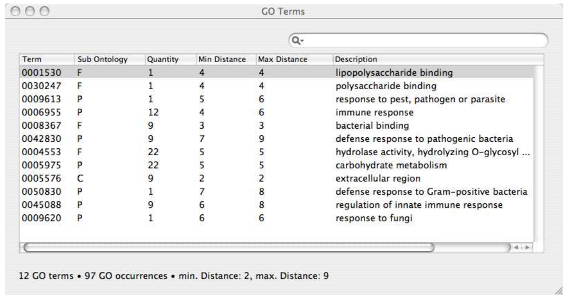
```
(BGBP1_DROME:0.651857,((Q8IQT7_DROME:0.000001,BGBP2_DROME:0.000001):0.000001,Q8IQT6_DROME:0.000001):0.738868,(((((((Q9U0G4_PACLE:0.238132,
(((Q8N0N3_PENMO:0.026599,Q8MVS9_LITST:0.073129):0.049706,Q68Y90_PENJP:0.042208):0.143487,Q70EW4_HOMGA:0.144823):0.089858):0.188524,
((Q95VY3_LUMTE:0.043142,O77072_EISFO:0.061421):0.481280,(Q7Z0T2_SPISA:0.524293,Q5I6N3_PATYE:0.388956):0.310459):0.285349):
0.105179,Q26660_STRPU:0.428058):0.093990,Q5W4Y9_SUBDO:0.639905):0.120727,(((Q8T9V2_AEDAE:0.075049,Q5MIY7_AEDAL:0.093191):
```



```
CLUSTAL W (1.82) multiple sequence alignment


Q8MVS9_LITST      ---------------MKTVLLSLSRMKGFVASVVLLACGALAADIVQP------------
Q8N0N3_PENMO      ------------------------MKGFVASVVLLACGALAADIVEP------------
Q68Y90_PENJP      ------------------------MKAIVAGVVLLACGVFAADVVDP------------
Q70EW4_HOMGA      --------------------------MMMLCLLLLACGVFAANVVDP------------
Q9U0G4_PACLE      --------------------------MRALCFLLLACGALAVDVLDP------------
Q26660_STRPU      ---------------------MVDLRHVLALLPVILVSINAYDVKNPEISLLTPRGIRF
O17492_ANOGA      --------------------------MQLVRVQLLIGSLLLATVGDWV-----------
Q7PQA4_ANOGA      --------------------------MQLVRVRLLIGSLLLATVGDWV-----------
Q7QCT6_ANOGA      ----------------------------MKRISVPLATLLLVGL---------------
Q7QCT7_ANOGA      -------------------------------MVLLGGVVFGVLL---------------
Q5MIY7_AEDAL      -------------------------MKGQEKVVLTVVWLLLCVTEL-------------
Q8T9V2_AEDAE      -------------------------MIQTQKRVVGTIGLLFCVVGL-------------
Q7QA32_ANOGA      ------------------------MHLSMGKLSILLVGLAMAGAVLG------------
Q5W4Y9_SUBDO      --------------------------MALQLYSLAAAGVILLSC---------------
O77072_EISFO      ------------------------------------------------------------
Q95VY3_LUMTE      ------------------------------------------------------------
Q5I6N3_PATYE      ------------------------------------------------------------
Q7Z0T2_SPISA      --------------------------MLLILLFGLVASALAVNIPQPKLTHLSTGDIQL
Q8IQT6_DROME      --------------------------------------------------------MFQI
Q8IQT7_DROME      ---------------------------------MLKDEGFEVSIPDEPGIQRVFYMFQI
BGBP2_DROME       -MRWEFLPCLLLLISNNKIFG----FKVPSINFEMLKDEGFEVSIPDEPGIQRVFYMFQI
BGBP1_DROME       --MPGLCIGILLLIGFGCTTA----YKIPTP-TVELLETGFSVSIPDEEGVKVVAFNVNR
Q5IY19_9DIPT      -MRHLLLPLVIVFLSAGISYGKIKQFSIPDVEIEFLKPRGFRASIPGVPGIKLFAFHARI
Q7QIB2_ANOGA      -MRWTWG-LLLFFVGQTVAY------TIPALRFEYPTMRGFRASIPDTPGLQMFAFHARL
BGBP1_MANSE       --MLKSVFVLFLVNY----LNSVRCLEVPDAKLEAIYPKGLRVSIPDD-GYTLFAFHGKL
BGBP_BOMMO        --MYKTCVWVLLF-------KIVLCYEAPPATLEAIHPKGLRVSVPDE-GFSLFAFHGKL
BGBP2_MANSE       --MWIKSVCLFATIAGCLGQ-RGGPYKVPDAKLEAIYPKGLRVSVPDD-GYSLFAFHGKL
BGBP_PLOIN        --MFVTFICFLACLTCSYGQPRAQQYVVPSAKLEAIYPKGLRVSIPDD-GFSLFAFHGKL
BGBP3_DROME       MADALRFVAWSCCLQLLFLLLGVQGYEVPKAKIDVFYPKGFEVSIPDEEGITLFAFHGKL
BGBP_TENMO        --MKVLVVFIFCLVR-----STFGQFEVPDALVEVFRPRGLRVSIPDQEGIKLFAFHGKI
BGBP_HYPCU        -RCARVCAVLFLFIQ-----ISYGQYQVPQVTVQALKPRGFKASIPDSPSVSLFVFQGNI
Q17233_BOMMO      --MGGRVLCLILFIK-----ISY--AQMPDVKIQAFRPKGLRISVQDVPKMTLFAFQGNL


Q8MVS9_LITST      ------------------------------------------------------------
Q8N0N3_PENMO      ------------------------------------------------------------
Q68Y90_PENJP      ------------------------------------------------------------
Q70EW4_HOMGA      ------------------------------------------------------------
Q9U0G4_PACLE      ------------------------------------------------------------
Q26660_STRPU      AYPDESGTTLVAFHYNINTPLSGVGAGQYNYDVTTTTDEYFVHENRDVDVENGDVVYYWV
O17492_ANOGA      ------------------------------------------------------------
```



Term	Sub Ontology	Quantity	Min Distance	Max Distance	Description
0001530	F	1	4	4	lipopolysaccharide binding
0030247	F	1	4	4	polysaccharide binding
0009613	P	1	5	6	response to pest, pathogen or parasite
0006955	P	12	4	6	immune response
0008367	F	9	3	3	bacterial binding
0042830	P	9	7	9	defense response to pathogenic bacteria
0004553	F	22	5	5	hydrolase activity, hydrolyzing O-glycosyl ...
0005975	P	22	5	5	carbohydrate metabolism
0005576	C	9	2	2	extracellular region
0050830	P	1	7	8	defense response to Gram-positive bacteria
0045088	P	9	6	8	regulation of innate immune response
0009620	P	1	6	6	response to fungi

12 GO terms • 97 GO occurrences • min. Distance: 2, max. Distance: 9

Figure 3.11: Invhogen graphical interface. When a user selects a GF in the window on the top left, definitions of all GF members are displayed on the right side of the main window. Furthermore, the protein MSA, the phylogenetic tree, and the GO term annotations are shown in the other three windows.

3.5 Discussion

INVHOGEN allows rapid selection of GFs according to various criteria. First, one can select homologous sequences for a user-defined set of taxa. The graphical interface provides easy access to all the data (MSAs, phylogenetic trees, taxonomic data, sequence annotations, GO annotations) required to interpret homology relationships between genes. Thus, INVHOGEN is a useful tool for comparative genomics, phylogeny or molecular evolutionary studies for invertebrates.

In the process of analyzing animal phylogenetic relationships, the approach used to structure the available vertebrate sequence data in a database (HOVERGEN) has been extended by a collection of all available invertebrate sequences. This work shows that under the assumption of 1.1 million known animals (97% of them are invertebrates) [69] only a small number of invertebrate species have proteins sequenced — and within these species, a dozen contribute the majority of the invertebrate sequences to this database. INVHOGEN has been built in the same way as HOVERGEN (to have a starting point for comparative analysis).

3.5.1 Other Approaches to Build Gene Families

It is also noted, that further work is needed to define homologous GFs. Different approaches exist that have not yet been fully exploited for the applications suggested here. Some approaches classify proteins into families using structural similarities [70] (structures available in Protein Data Bank (PDB) [71]), or grouping them into families on the basis of the presence of shared domains or similar domain architecture [72] using domain databases like Prodom [73], Pfam [74], and InterPro [75].

Apart from classification methods based on sequence alignments and motifs, statistical learning methods applying support vector machines (SVM) [76] are useful for classifying diverse protein sequences. SVM and related approaches will complement sequence similarity and clustering methods. Another approach is adopted by ontology-driven systems to build families of specific proteins [77]. Ontologies are also useful for pre-processing BLAST searches presenting a weighted list of GO terms associated with similar sequences to give information about potential functions of unknown proteins.

However, it remains to be seen how approaches like OntoBlast [5] can be utilized to reconstruct more reliable GFs. We hope that more sophisticated algorithms using all available methods will substantially reduce the number of GFs with only very few members (Table 3.1). Additionally the discrepancy between few often sequenced species and many infrequent sequenced species should be kept in mind when generating GFs. Moreover sequence sampling is biased towards a few model organisms. This may also be the reason for a lot of GFs with few species. Thus for a better understanding of the evolution of GFs one should sequence genes from a wide variety of taxa and not only from a few well-known model organisms.

3.5.2 Annotation Problems

Besides the unbalanced distribution of the different organisms in INVHOGEN, two other annotation issues arose in the course of building the invertebrate GFs: (i) inconsistent nomenclature for protein sequence descriptions, and (ii) the lack and the (sometimes poor) quality of GO terms annotations.

Gene Product Descriptions

The issues of consistent nomenclature apply to gene product descriptions as well as organismal taxonomy [78]. For database queries to be meaningful it is important that consistent names are used for the same source of data. It is not uncommon for the same gene to have several different names in the sequence databases due to synonym terms, acronyms, morphological and derivational variants, reinforced by the ad hoc use of orthography such as capital, spaces, punctuation (e. g. NF-Kappa B, NF Kappa B, NFkappaB and NF kappa B) and the inconsistent naming conventions (e. g. IL-2 has many variants such as IL2, Interleukin 2 and interleukin-2). On the other hand, names and their acronyms are often classified in different semantic classes, depending on a given context.

For other kinds of data similar issues arise. Morphological data tends to be described in an idiosyncratic fashion, although efforts are being made to standardize nomenclature (Structure of Descriptive Data subgroup)[1].

These inconsistencies for gene product names often were a challenge for naming GFs (described in Section 3.2.2). It remains also a problem in a very common task in information extraction, *named entity recognition*, where named expressions (such as protein names, chemical compounds, diseases, symptoms etc.) have to be recognized and classified.

GO Annotations

Analysis of how GO terms are used to annotate SWISS-PROT/TrEMBL reveals that much of GO is either barely used or not used at all: As it is shown in the result section every fourth sequence entry has no GO annotation (Table 3.5, Figure 3.3), and most sequence entries are only annotated with few GO terms which appear by reason of their general classification in many other GFs, e. g. nucleus in 1,549 GFs (Table 3.8). By this it means that a large quantity of proteins are 'under-annotated', so a general term has been assigned when a more specific term would be better. Figure 3.6 to Figure 3.8 point out the distribution of all sub-ontology terms in Invhogen. 352 different annotated cellular component GO terms from 1,695 cellular component terms in GO (20.8%) are used in all sequence entries. Out of them only ten terms are presented more than 1,000 times, 195 terms are between four and 97 times, and 123 less or equal three times. Both other diagrams in Figure 3.7 and Figure 3.8 have the same characteristics of term distribution: 15.7% of all molecular function sub-ontology terms are annotated and 16.3% of all biological process sub-ontology terms, again with only a dozen of highly distributed GO terms not far distant from the root GO term entry.

Reasons for the large number of weak (Table 3.7) and/or general annotations from the researcher's perspective — besides automatic approaches to annotate sequence entries with GO terms — are the lack of biological knowledge, an incomplete understanding

[1] `http://www.tdwg.org/sddhome.html`

to match gene information with especially more specific GO terms, or mis-annotations (e. g. due to spelling mistakes). As a consequence it might currently not be meaningful to use annotated information from GO terms in protein sequence entries as additional criterion to approve the relationships of sequences within each GF. However, for some but not many small GFs there is a kind of 'semantic' support regarded from consistent and specific GO term annotations.

Methods of Resolution In order to overcome the difficulties to better annotate anonymous sequences with the GO vocabularies many tools and software programs have been developed through an automatically or manually curated search for the associations between sequence entries and GO terms [5, 79, 80, 81]. Other programs are more concentrated on visualizing, comparing, and plotting GO annotation results to find the differences or anything new in sequence datasets [82, 83, 84].

The large-scale assignment of GO terms to SWISS-PROT/TrEMBL proteins using electronic methods is a fast and efficient way of associating high-level terms to a large number of proteins. To provide more reliable and specific annotation, the GOA project [3] also makes use of manual curation using information extracted from published scientific literature. This process is slower than the use of electronic techniques but provides more accurate information as all annotation is validated by a team of skilled biologists.

Contribution to the GO project works similarly. One can send comments, questions or suggestions regarding GO to a so-called GO helpdesk or one can subscribe to a number of mailing lists to follow the development of different aspects of the GO project. To get in touch with the GO curators there exists the GO curator requests tracker which allows one to denote requests regarding annotation issues, bugs, or ideas to improve the project.

A drawback besides a lot of advantages of both contribution ways is that researchers are not directly involved in the development of the database and the ontology, respectively. One solution to address this kind of unilateral communication (in some way) between scientists might be to offer all of them a common web platform which also integrates *social network* aspects to enable knowledge elicitation for diverse domains and as its central part an application to cooperatively develop ontologies in terms of an ontology wiki. This Internet application called ONTOVERSE is described in the next chapter.

Chapter 4

Ontoverse

Ontoverse is a research project funded by the German Federal Ministry of Education and Research, that offers an approach within the Ministry's promotional focus on eScience and knowledge management while concentrating on life sciences as the domain of interest.

4.1 Introduction

Knowledge networking comprises two different aspects: collaborative knowledge management in communities (human networks) and effective information integration (data networks). Thus, on the one hand techniques are regarded that help to structure and interlink existing knowledge sources effectively with ontologies as the core technique. And on the other hand, in knowledge networking people share their knowledge via social networks.

The Ontoverse project aims at combining these two points of view: establishing a platform, that provides tools for designing ontologies for annotating and interlinking knowledge sources and that also helps people to build up social (scientific) networks. This platform is called the Ontoverse ontology wiki. It comprises support for collaborative ontology engineering, an ontology based publication management system and solutions for knowledge exchange in scientific communities.

4.2 The Need for Collaborative Ontology Development

Recently, the optimization of storing, retrieving and integrating data is becoming a popular focus for the WWW in general and a fundamental task for scientific contexts. For the focused domain of interest, the life sciences, the particular problem is the integration of heterogeneous data. For example, bioinformatics data not only consist of customary textual items (scientific publications), but also comprise nucleotide sequences, amino acid sequences, 3D structures of molecules and a manifold of other experimental results. Such diverse biodata need to be stored and structured effectively. Recent progress in the life sciences has already led to the accumulation of biodata that now demand classification, accessibility and visualization.

4.2.1 Representing a Shared View

A shared understanding of a domain is needed as a basis for scientific discussions. If no consensus on a domain of knowledge and its key components exists, definitions have to be mentioned in every single discourse on that topic. Thus, ontologies form the basis for communication within a community as well as for human-machine interaction [85]. It is most desirable to have ontologies produced collaboratively by a large community, to ensure that they do indeed represent a shared view. Opinions and suggestions of a broad community of domain experts should be regarded.

4.2.2 Information Integration for Scientific Data

Additionally to the management of the vast amount of biodata there is also an urgency to collect scientific cognitions and make them multidisciplinarily accessible. GO for example structures investigated genes and information about them (Section 2.2.1). This addresses a main problem in genetics which is the multiple denotation of similar genes which were found in different organisms. This problem leads back to the rapid increase of knowledge. Scientists often specialize on single research domains, for most of them it is hard to keep track of all new developments even in these limited areas, even harder it is to recognize trends in other fields that might effect ones own research. Because of these advantages a growing interest in ontologies can be noticed especially in the bioscience community, resulting in different ontology projects GO and the National Center for Biomedical Ontology [86] which includes the Open Biomedical Ontologies.

4.2.3 Experiences in Developing a BioInformatics Ontology for Tools and Methods

Within the Ontoverse project an ontology, called BIO2Me, to gather experiences is designed which are incorporated into the development of the platform [87]. For that purpose the domain of bioinformatics tools has been chosen. Bioinformatics provides tools for the efficient processing of experimental data (e. g. from genome sequencing), sequence analysis, and structure prediction and visualization (amongst others) and by now there is a variety of tools available which handle similar biological questions with partly completely different computational methods. Without a structured overview, even for experts in bioinformatics it is hard to decide which tool fits their individual requirements best. Therefore, the knowledge about these tools have to be collected and organized, for otherwise the full potential of existing tools is not be utilized. The goal of BIO2Me is to provide information about certain bioinformatics tools and methods, and to enable search for these tools within the ontology that meet the user's needs.

The domain of BIO2Me eminently points out the need for collaborative ontology engineering. To represent bioinformatics tools with their applications, the whole bioinformatics research field and biology itself have to be displayed adequately in a structured way, as the application range of every tool has to be set into this context.

4.3 Editing and Maintaining Ontologies

Ontology engineering is a process of several phases; mainly a conceptualization phase, a phase of actual editing the structural data, a maintenance phase that includes updates, corrections and ontology enrichment, and as an optional last phase the reuse of the ontology (other divisions are also possible, e. g. in [88]).

Every ontology project begins with a conceptualization, where basically the aim and scope of the ontology as well as design guidelines are defined. It is useful to provide an Ontology Requirement Specification Document (ORSD) [89] for that purpose, that can be filled with the respective decisions. During this conceptualization phase, thematic discussions of a domain community are extremely useful to capture the characteristics of the domain.

While the conceptual phase forms the shared basis for further collaborative work, the following steps of the engineering process profit from a vital community. *Social tagging* and *knowledge elicitation* systems are integrated to collect suggestions for further concepts. Even after a stable state is reached, ontologies kept in community environments are furthermore be permanently maintained and updated.

It requires a lot of coordination, communication and support by adjusted tools to enable such collaborative development of ontologies, especially in early phases when the basic structure of the ontology is planned. Another major role for an interested community is to interlink and classify current ontology projects to make them accessible and traceable. This is a necessary condition to enable effective ontology reuse and to prevent redundant projects.

A necessary element of supportive technology is a system to make all changes and edits easily visible and traceable. This also includes an option for immediate notification when changes on the ontology have been made. Further requirements for collaborative work are discussed in [90].

Das *et al.* [91] provide an evaluation of some ontology editors with regard to multi-user collaboration. A number of editors somewhat support this aspect and the web application WebOnto [92] exceeds the requirements. But WebOnto is no longer under development and the other multi-user editors like Chimaera [93] and Ontolingua [94], also realized as web applications, are not available anymore. By now a new generation of collaborative approaches has emerged, e. g. [95, 96, 97, 98], who all have slightly different foci and aims, but have not yet implemented a running system.

4.4 Ontology Wiki

The Ontoverse approach implements an ontology wiki; i. e. an editor platform that supports distributed work on structural (ontological) data as well as informal discussions and annotations (proto-ontological data). Interested users can view and use ontologies, join existing ontology projects or plan and start a project anew. All different phases of ontology development like conceptualization, editing, maintenance and reuse are supported (see [91] for a survey of published ontology design methodologies). This also connotes, that the ontology wiki is a platform for multiple ontology projects that have to be administered diligently and provided in an easily accessible way.

4.4.1 User Community and Collaboration

Some approaches have been made to support collaborative ontology engineering e. g. in [95, 99]. What is new in Ontoverse, is the explicit support of a social network closely combined with a Web-based ontology editor. A focus is placed on the support of a heterogeneous community. Potential users differ in their fields of interest and skills: On the one hand knowledge and expertise is needed from domain experts (DEs). On the other hand ontology languages can only be fully exploited by ontology designers (ODs).

4.4.2 Key Aspects

Major innovations in Ontoverse are:

- An open collaborative approach that takes into account all the people who have expert knowledge in a certain knowledge area. The system brings together ODs and DEs and regard their different states of knowledge. Within such an approach to community collaboration, certain considerations on roles and their rights are necessary as discussed in the last paragraph of Section 4.5.1.
- All different phases of ontology development are supported. The wiki encourages thematic discussions and the adding of unstructured (proto-ontological) data in the actual ontology editing process.

- Integrating the publication management system PubDB to store and manage thematic documents and adding information extraction functionalities (Section 4.6.13). Scientific publications are one kind of knowledge source to gather information about a domain. Within the Ontoverse project a publication database is used as the knowledge source for an IE application that supports the community in identifying relevant concepts, instances and relations from texts, which can then be incorporated into the ontologies. In return, the newly developed ontologies themselves help to retrieve relevant documents from publication databases. Ontoverse members can also tag arbitrary publications with their own keywords. These tags are mapped whenever possible with the concepts, relations and individuals in the ontologies. Doing this the platform facilitates social search and DE identification as described below.

- The identification of DEs whenever needed by the OD during the editing process is supported. Registered DEs can define expertise in their profiles, which are then used to identify fitting DEs in cases of support requests by ODs.

- A project's ontology is subject to successional changes. To enable periodical consistent and stable releases the system incorporates a release process. If project members decide that it is time for a new release the current state of the ontology is copied into a debugging branch (Figure 4.1).

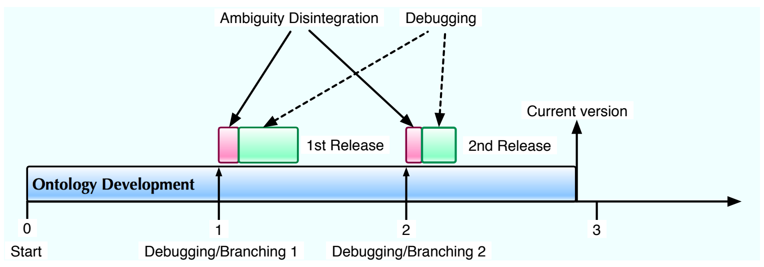


Figure 4.1: Ontology debugging and release process. After certain time intervals an ontology should be submitted to a disambiguity resolution process followed by a debugging stage where inconsistencies are removed. The final stable ontology gets a fixed release number.

4.5 Challenges and Tasks of Collaborative Ontology Development with Ontoverse

Along with the benefits of collaborative ontology editing some new challenges emerge. In order to identify the most important ones they are divided into conceptual and technical categories. Nevertheless, because of the strong interdependencies between them this separation remains to some degree artificial.

4.5.1 Conceptual and Process Challenges and Tasks

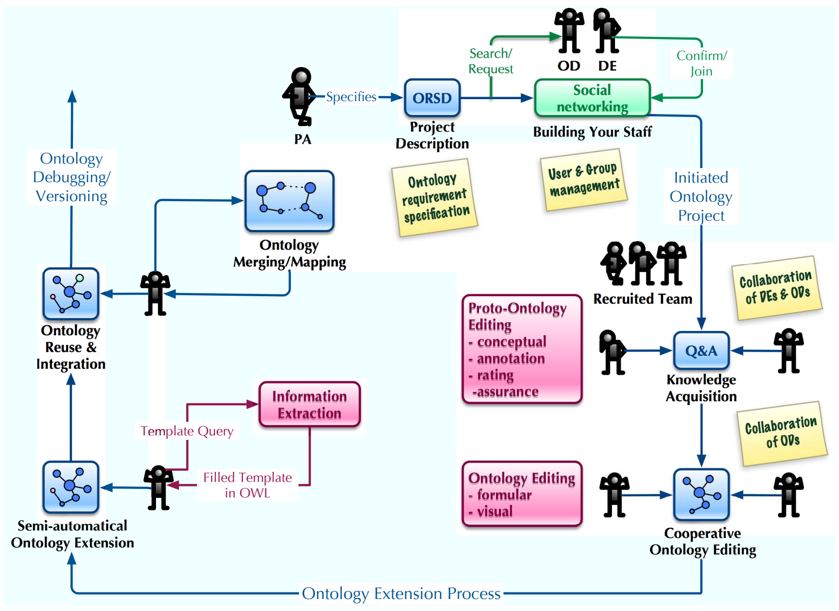


Figure 4.2: Schematic illustration of the Ontoverse ontology life-cycle process.

To offer an integrated ontology development platform that directly connects DEs with ODs the Ontoverse project has to address important conceptual challenges. Particular objectives are the proper integration of both the inter-ontological (i. e. ontology mapping and merging) and the intra-ontological (concerning the concepts and properties inside one single ontology) data processing level, the connection of proto-ontological (i. e. informal and semi-formal) and ontological (formal) data and the adequate management of the community. Figure 4.2 illustrates the ONTOVERSE ontology life-cycle containing the conceptual challenges that are discussed in the following.

Inter-Ontological and Intra-Ontological Level in Ontology Engineering

The assembling of ontological data can be conceptually divided into inter- and intra-ontological stages. The intra-ontological level has to support tools and methods for building up a knowledge base from scratch or by extension of a given ontology with further concepts and relations. In Figure 4.2 the process steps *Cooperative Ontology Editing* and *Semi-automatical Ontology Extension* belong to the intra-ontological engineering level. The primary actors in these workflow steps are the ODs and an IE module for (semi)-automatical ontology extension. The IE screens textual data to fill predefined templates with facts that can be used to extend the knowledge base after being curated by ODs. To ensure the correctness and quality of the ontology the designers can consult DEs via synchronous and asynchronous communication channels. The process step *Knowledge Acquisition* reflects this in Figure 4.2.

Conceptual Challenges

Classes vs. Instances One important design issue is whether a concept should be represented as a class or an instance. In general, there is no clear division between classes and instances, and a concept can be either one. Choosing between them represents a design choice that is dependent on the purpose of the ontology [100].

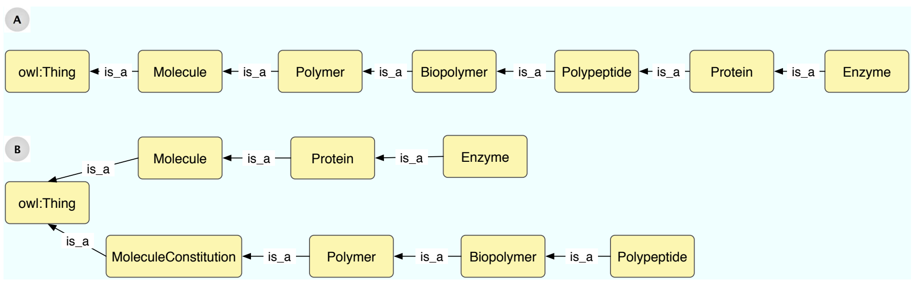


Figure 4.3: Vertical vs. horizontal hierarchy construction.

Vertical vs. horizontal hierarchies The class vs. instances design issue is closely connected to the vertical vs. horizontal construction of class hierarchies. Particularly ontology design novices tend to model concepts in deep vertical hierarchies (Figure 4.3, A). While such design can be technically correct in principle — this is not true for this example by the way as proteins can also consist of more than one polypeptide — it has always to be checked whether they are appropriate, too. In many cases it would be better to identify properties of the concepts that allow to use the horizontal dimension as well. The example could also be reorganized into a more horizontal structure (Figure 4.3, B).

Identification of corresponding concepts and properties To map or align ontologies the corresponding concepts and properties first have to be identified. Problems occur when semantically similar concepts are using labels that cannot be identified through pattern matching methods. In these cases correlating concepts have to be identified and annotated manually. Another kind of aligning/mapping problems are semantically asymmetrical correlations. Such kind of correlation exists, if in one ontology a single concept and in another ontology a group of connected concepts represent the same section of the domain. It can get even more complicated if the same domain knowledge has been modelled in two largely different ways. The challenge in both cases is to narrow down the relevant properties and related concepts.

Merging equivalent or semi-equivalent sub-structures After the identification of the corresponding sub-structures in a merging project ODs have to decide how two sub-structures could be merged the best way.

Proto-Ontological and Ontological Data

Ontoverse introduces a distinction of proto-ontological and ontological data, which are both handled in the same tool. Ontological data refers to formalizations of ontological elements. In contrast, proto-ontological data can be every piece of information not yet being brought into formal structure. This ranges from the ORSD (Section 4.3), over semi-formal concept graphs to the rating and annotation of concepts and relations (Figure 4.2 steps *Project description* and *Proto-Ontology Editing*).

Managing the Community: Roles and Rights

From the user's perspective, ONTOVERSE offers role management with different access privileges. On the part of ontology development each registered user is capable of initiating a new project or to participate in existing ontology projects. Typically any collaborative project should consist of several users with knowledge in ontology design and/or the specific domain it was originally created for.

Figure 4.4 points out the natural structure of ontology projects in an organization chart. Apart from the organization of the ONTOVERSE platform itself the ontology building part is focussed on the interrelation between DEs and ODs. Every project is administrated by a project administrator (PA), who is monitoring all the changes and is responsible for conflict resolution within the ontology.

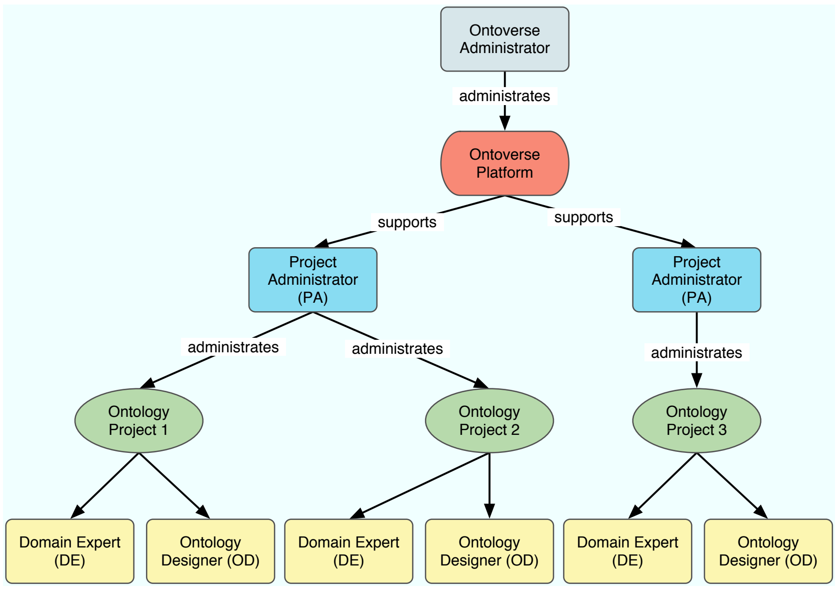


Figure 4.4: Every ontology project is administrated by a single group of PAs composed of one or more members. Customarily both DEs and ODs are involved in ontology development.

Access rights are coupled to individual roles of specific projects. That means a user can act as DE in one project and as OD or even PA in another. This comprises the possibility for one user to play different roles in one project.

Ontology designers are mainly entrusted with the translation of domain knowledge into a formal ontological representation (or transforming proto-ontological into ontological data). This stage is established by a dialog with human experts in order to elicit knowledge. In the following stage the OD codes the knowledge explicitly in the ontology. This process iterates until the ontological coding is judged to be satisfactory by the experts.

However, one difficulty in communication between ODs and DEs is that the knowledge engineer does not always know the technical terms of the DE. Another problem may be in expressing the knowledge in explicit terms. To overcome these difficulties the support of proto-ontological data as modes of expression is needed. Functions of the different user groups are summed up as follows:

Domain Experts:

- Collect knowledge, provide it informally (proto-ontological data) and discuss it. Contribute new scientific findings.
- Collect publications relevant to actual topics of interest and tag them.
- Answer domain-specific questions posed by ODs.
- Discuss and benchmark existing ontologies.

Ontology Designers:

- Exploit the input contributed by DEs and transfer it into formal ontology languages (ontological data).
- Discuss formal knowledge representations, edit and maintain them.

Project Administrators:

- Define aim and domain of a new ontology.
- Coordinate discussions and monitor the collaboration process.
- Release versions of an ontology.

Considering the user roles and potential workflows as well as general problems in collaborative systems, the following requirements for the platform can be derived.

- Elaborated communication channels, adjusted for discussions DE-DE, OD-OD and DE-OD.
- User profiles and user networks: show fields of expertise, search for experts.
- Information on authorships, copyright guidelines.
- Collaboration guidelines and principles of ontology engineering.
- Community awareness features, tracing of changes.

4.5.2 Technical Challenges and Tasks

Besides social and interpersonal aspects, offering a system for cooperative work of large user groups addresses some technical problems to resolve in order to provide a platform on the Internet for designing ontologies in a collaborative way. Because of its inner structure, RDF data is typically large-scale, highly interconnected and heterogeneous in contrast to relational data. Thus, it is indispensable to manage such datasets cleverly to allow cooperative development. Furthermore, the system should support graph-based navigation, querying facilities, and editing functionality accompanying ontology generating processes.

Ontology Storage and Managing

Due to the fact that the requirements for storing RDF data efficiently are different from relational databases two approaches exist to store and manage RDF data. The first approach is based on relational databases which store RDF data in a persistent data model by mapping the RDF model to the relational model. On the other hand there are systems that implement a native store with their own index structure for RDF triples like Jena2 [101], Redland [102], and YARS [103]. To record changes in an ontology, a network equivalent of a shared 'blackboard' was needed. Remote programs can place objects on the board, examine these objects or remove them. This is generally realized by a data structure called a tuple space that is optimized for distributed programming [104]. The implementation of this back-end part in Ontoverse is described in Section 4.6.14.

Visualization

The exploration of large datasets is generally an important but difficult problem. Visualization techniques are useful in solving this problem. Data types to be visualized for ontologies are hierarchies and graphs [105]. These data records have many relationships (connections) to other objects (called nodes) and build ordered, hierarchical, or arbitrary networks of relations representing such interdependencies. Ontology visualization also needs to be tightly integrated with the Ontoverse system used to deal with the vast amounts of information. The aim is to bring the benefits of visualization technology to the user to allow a better, faster, and more intuitive handling of huge ontological data. In viewing or editing large ontologies, one first needs to get an overview of the graph structure. In the overview, the user is able to identify interesting regions to edit or explore subgraphs within the ontology which are currently processed by other users.

Interaction techniques allow the user to directly interact with the graph and dynamically change the visualizations, as well as select subsets of the data for further operations. These techniques can be categorized based on the effects they have on the display. Navigation techniques focus on modifying the projection of the data to smoothly navigate through the ontology onto the screen. View enhancement methods like zooming or distortion [106] allow to adjust the level of detail on part or all of the visualization, modify the mapping to emphasize some subset of the data, or collapse unrequested data. Selection techniques provide users with the ability to select interesting parts of the ontology for operations such as highlighting or filtering.

In addition collaborative aspects into the visualization are incorporated. That is why e. g. locked parts of the ontology are particularly marked in the graphs and changes are notified. Additionally the view depends on the user role. The realization of ontology editing and visualization in Ontoverse is also given in Section 4.6.14.

4.6 Ontology Wiki Architecture

4.6.1 Overview

From these goals in the aforementioned sections, a high-level list of requirements for the ontology wiki is shown in the following and summarized in Figure 4.5:

- A system to allow users to create user accounts and add profiles about themselves: This requires them to log in with a username and a password (Section 4.6.2, Page 75).
- News blog: This allows editors of the site to create news reports and publish them on the front page (Section 4.6.3, Page 81).
- Discussion forum system: Forum moderators are able to create a number of forums in which users can create new topics. Each topic can have any number of posts (Section 4.6.4, Page 83).
- Blogging engine: This allows users to create their own blogs about their projects and development experiences. It allows users to post blog entries using desktop blogging clients as well as the Web (Section 4.6.5, Page 86).
- Photo gallery for each user of the site: This allows users to upload photos to their profiles or to relevant projects (Section 4.6.6, Page 90).
- E-mail newsletter: The newsletter can be sent to all members of the site that opt in to receiving e-mails from the site (Section 4.6.7, Page 91).
- Friendships system: This system allow users to add other users to a friends lists or other types of relationship (Section 4.6.8, Page 93).
- Tagging and searching support: Projects, photos, and publications are taggable to make it very easy for users to search and browse these objects (Section 4.6.9, Page 95).

- Google Maps integration: Opens up the possibilities of embedding interactive, scrollable maps for project members (Section 4.6.10, Page 98).
- Ontology projects and project's wiki: The ontology building part of the ontology wiki to manage and maintain ontology projects (Section 4.6.11, Page 100 and Section 4.6.12, Page 102).
- Scientific publication database PubDB: The source for project-specific document collections mainly to extract information (Section 4.6.13, Page 106).

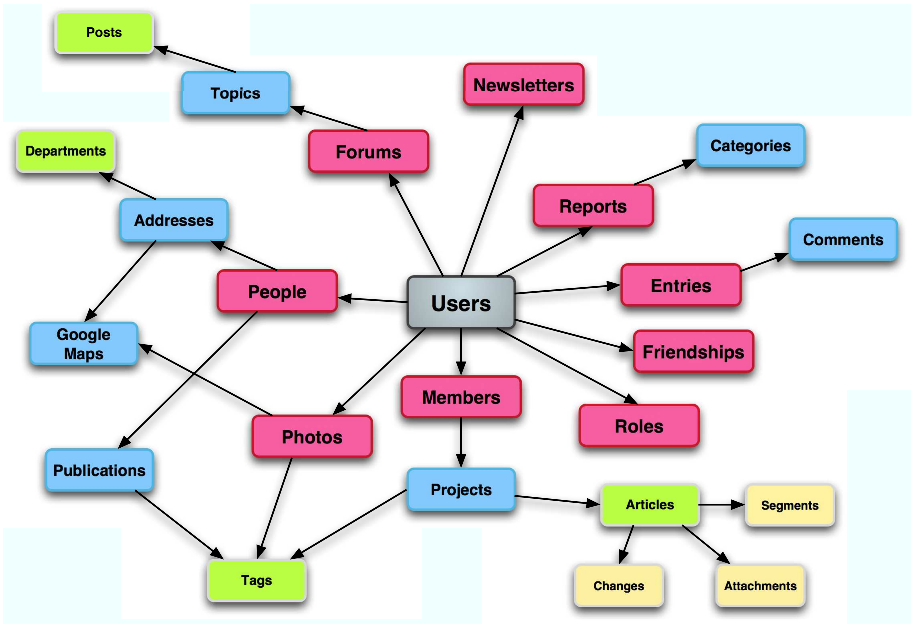


Figure 4.5: Overview of the architectural components in the ONTOVERSE ontology wiki from a user's perspective (see also Table A.1).

4.6.2 User Management System

In this section, the user management system with user accounts and a role-based group system is described. This allows users to create accounts and log in to the site. The ONTOVERSE administrator (OA) role maintains control to regulate who can administer the

ONTOVERSE platform. A web interface allows administrators to manage the permissions of each user, including disabling accounts.

User Model

The USER model holds the account information. It defines the information that will be stored about the users and how a user's input can be validated on sign-up. For the log-in process a user needs a password. This password is stored in the user database table using a one-way hashing algorithm [107]. By storing the hashed value of the user's password in the database, one can check that the user has entered the correct password by calculating the hash of the entered password and comparing that to the hashed value stored in the database. The required database fields for the USERS table are shown in Table 4.1 (some internal table fields are skipped in this table).

Name	Type	Description
id	integer	The primary key
login	string	A unique name that the user will use as a log-in name
email	string	The e-mail address is required for the confirmation process after a user has signed up
crypted_password	string	This will store the hash of the entered password
enabled	boolean	If a user wants to remove the account, it needs to be disabled
created_at	datetime	The date and time that the user was created
updated_at	datetime	The date and time that the user was last updated

Table 4.1: USERS database table.

Person Model

In contrast to the USER model, the PERSON model holds the personal information for completing a user's profile with personal data (title, name, gender, home page, and so on). Table 4.2 shows the complete structure of the PERSON model's database.

One source for this kind of data might be extracted from publications (for more details see Section 4.6.13).

Name	Type	Description
id	integer	The primary key
gender	string	The gender of a person
title	string	An academic title of a person
firstname	string	First name of a person
middlename	string	Middle name of a person
lastname	string	Last name of a person
birthday	date	A person's birthday
initials	string	Initials of a person
email1	string	First e-mail address
email2	string	Alternative e-mail address
homepage	string	The person's home page

Table 4.2: People database table.

Address Model

In addition to the Person model, the Address model represents address specific data. Addresses table and People table are linked in a one-to-many relationship. The required database fields for the Addresses table are shown in Table 4.3.

Name	Type	Description
id	integer	The primary key
street	string	Street
city	string	City
zip	integer	Zip code
state	string	State
country	string	Country
add_info	string	Additional information (e. g. building)
lat	string	Latitude
lng	string	Longitude

Table 4.3: Addresses database table.

Department Model

The Department model contains information about a department's name and organization, a description, and an entry for its home page. This model is linked in a many-to-many relationship with the ADDRESS model and PERSON model, respectively. The relationship between department and person is complemented by a history function storing all previous departments of a single person (Figure 4.7). The required database fields for the DEPARTMENTS table are shown in Table 4.4.

Name	**Type**	**Description**
id	integer	The primary key
name	string	Name of the department
organization	string	The department's organization
description	text	Description of the department
homepage	string	The department's home page

Table 4.4: DEPARTMENTS database table.

Account Management

The authentication system offers the following features required for authenticate users onto the platform (Figure 4.6):

1. Sign up: In the sign-up form a user gives information about his profile.
2. Activation: He is being sent an e-mail to ensure he had given a legitimate e-mail address.
3. Logging in: Users without an account first have to sign up for an account. If a user logs in successfully he is redirected to the initial URL he entered before the log-in page was invoked, basically the start page.
4. Failed log-in: An invalid log-in is redirected to the log-in form.
5. Reset password: A user has forgotten his password to the system. He clicks a link to reset the password, and the system sends a link to create a new one by e-mail.

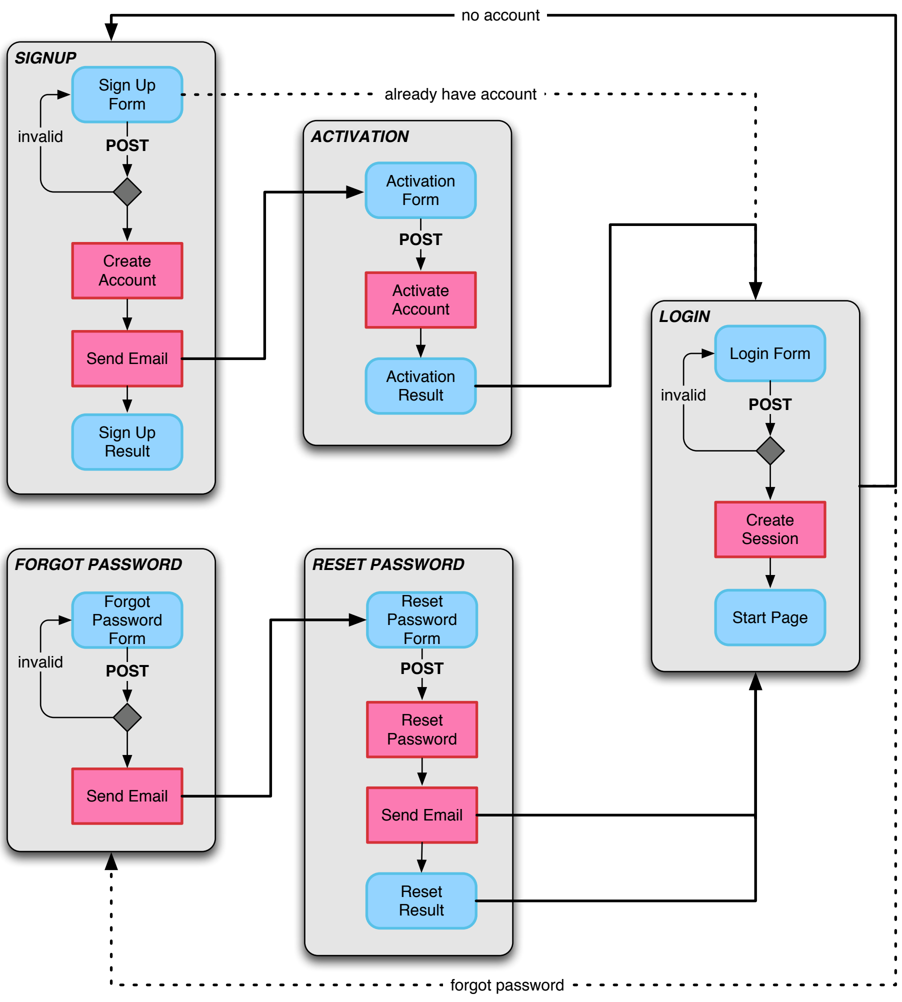


Figure 4.6: Authentication system in ONTOVERSE.

Role Model

In order to assign different permissions to different users, a ROLE model is created to store the different roles of users on the site. This will not store what the role is capable of, just the name (and description) of the role. The actual capabilities or restrictions of each role is defined in the code. The required fields of the ROLES table are shown in Table 4.5.

Name	Type	Description
id	integer	The primary key
name	string	Name of the role
authorizable_type	string	Authorizable type
authorizable_id	integer	Authorizable ID
created_at	datetime	The date and time that the role was created
updated_at	datetime	The date and time that the role was last updated

Table 4.5: ROLES database table.

USER and ROLE models are linked together using a join table. This join table simply stores the IDs of the two separate models and links them together. On the model site, a many-to-many relationship is specified by stating that the USER model has and belongs to many roles, and the ROLE model has and belongs to many users (Figure 4.7).

Controllers

The authorization process decides whether a user is allowed access to some feature. It is distinct from the authentication process, which tries to confirm a user is authentic. Implementing this technology requires: a USERS controller, a SESSION controller, and an ADMIN controller.

Users Controller The USERS controller provides the usual REST methods for accessing the USER model: `index`, `show`, `new`, `create`, `edit`, `update`, and `destroy`. A user is not actually deleted, but only disabled by setting the *enabled* field to false. To allow

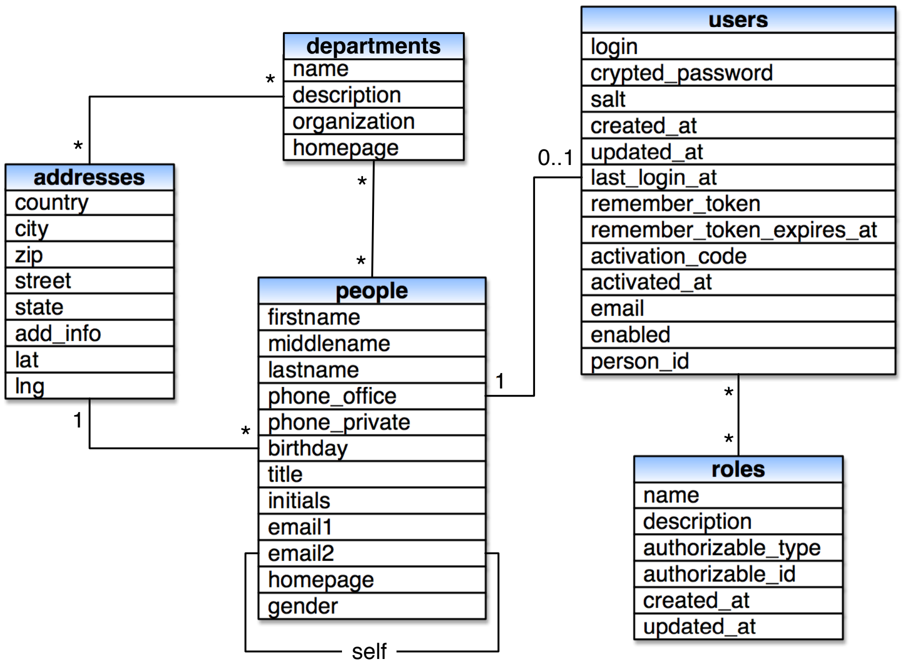


Figure 4.7: Entity relationship diagram for users, user's profiles, and roles.

to re-enable a user's account, administrator permissions are required. The activation functionality completes the signup procedure after user confirmation via e-mail.

Session Controller The SESSION controller needs the following methods to allow a person to join the site, log in, and log out:

- The `signup` method allows users to enter their details to become new members of the site.
- The `login` method checks e-mail address and passwords of the users to allow them to log in to the site.
- The `logout` method logs out users from the site.

Admin Controller Through this controller roles are defined by administrator for the whole platform (e. g. 'editor', 'moderator') and roles which are project-specific.

Other User Management System Extensions

- In the sign up form a security feature called CAPTCHA[1] requires user to type the letters shown in a distorted image, to catch a large portion of automated spam bots.
- 'Remember me' allows to remember the log-in status of its users. A user does not need to enter the log-in name and password each time visiting the site because the web browser remembers them.
- The sign up procedure integrates OpenID[2] into the user accounts system, allowing users to log in with their existing OpenID identity. This provides people an opportunity to sign on to multiple web sites using the same identity.
- After signing up successfully, Google Maps Geocoder[3] tries to find out the coordinates of the given address and stores them in the Addresses table if they are available.

4.6.3 Building a News Blog

A news blog is built to be shown at the Ontoverse front page to keep the users informed of developments at the site as well as keeping the site up to date. The basic functionality of the news list allows administrator users or editors to create news reports, along with an archive of them. Reports can be checked and edited by other editors before going published.

A new role, called editor, is provided to create and edit news. This allows OAs to give permissions to certain users to write and edit reports without giving them access to editing other parts of the site.

The news report functionality provides RSS[4] and Atom[5] feeds of the report, along

[1]http://en.wikipedia.org/wiki/Captcha
[2]http://en.wikipedia.org/wiki/Openid
[3]http://maps.google.com/
[4]http://en.wikipedia.org/wiki/RSS_(file_format)
[5]http://en.wikipedia.org/wiki/Atom_(standard)

with providing an API to the news feature. Both feeds let users to subscribe to the news feed with an RSS newsreader and be automatically notified when a new report is posted.

Report Model

The individual news report uses a model called REPORT. Reports can be created, checked and edited by administrators or editors. Editor users are able to add markup without to write HTML in the reports. Each report belongs to one category which can be created and maintained through a web interface. The necessary fields to implement its functionality are shown in Table 4.6.

Name	Type	Description
id	integer	The primary key
title	string	Title of the news report
synopsis	text	A short synopsis that will be shown in a list of reports
body	text	The text of the report itself
published	boolean	Reports can be saved and edited before being published
created_at	datetime	The date and time that the report was created
updated_at	datetime	The date and time that the report was last updated
published_at	datetime	The date and time that the report was published
category_id	integer	Category of a report
user_id	integer	ID of the user who created the report

Table 4.6: REPORTS database table.

Category Model

Each report belongs to one category, so a one-to-many relationship is defined — each category can have many reports. A category consists only of a name field limited to 80 characters. Figure 4.8 shows the relationships among news reports, categories, and users.

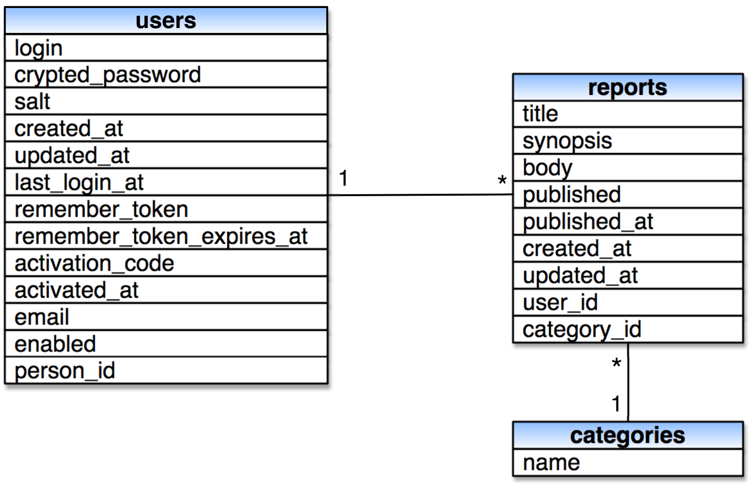


Figure 4.8: Entity relationship diagram for news reports, categories, and users.

Controllers

Report Controller The REPORT controller provides the normal REST CRUD actions. The `new`, `create`, `edit`, `update`, and `destroy` methods are needed for the user who has the editor permission. Since reports can belong to a category, they should be accessible by URLs such as `categories/1/reports`, which returns all of the reports with category ID 1. An index view shows a full list of all reports and links for editors to quickly add a new report or edit an existing one.

Categories Controller This controller allows users with the relevant permission to add, edit, or delete categories. A user can get a list of all categories together with the number of reports in each category. Clicking a category will take the user to the list of reports within that category.

4.6.4 Discussion Forum

In this section, the discussion forum for the ONTOVERSE community is presented. It enables users to discuss various aspect of general usages, project-specific themes, and other topics. Only OAs and other nonadministrative users, called moderator, are capable to create a number of forums. Within each of these forums, users can create topics. Each topic then has any number of posts within it about that topic.

The forum structure consists of three models: Forum, Topic, and Post. A forum has many topics, and a topic, many posts. In addition to this, each topic and each post belong to the user who created them. For performing reasons the Forum and Topic models make use of counter caches to speed up database queries.

Forum Model

The Forum model consists of a name and a description. Forums can be created, edited, and deleted only by a moderator or OA. A counter cache stores the number of topics per forum. Table 4.7 shows the complete structure of the Forum model's database.

Name	**Type**	**Description**
id	integer	The primary key
name	string	Forum name
description	text	Description of the forum
created_at	datetime	The date and time that the forum was created
updated_at	datetime	The date and time that the forum was last updated
topics_count	integer	The topic counter cache

Table 4.7: Forums database table.

Topic Model

The Topic model has the topic name along with the user ID of the user who created the topic. Any logged-in user can create a new topic, but only moderators (or OAs) can edit or delete topics. Deleting a topic will delete all of the posts within that topic. A counter cache stores the number of posts per topic. The Topic model's database structure is shown in Table 4.8.

Name	Type	Description
id	integer	The primary key
name	string	Subject of the topic
created_at	datetime	The date and time that the topic was created
updated_at	datetime	The date and time that the topic was last edited
posts_count	integer	The posts counter cache
forum_id	integer	ID of the forum that this topic belongs to
user_id	integer	ID of the user who created the topic

Table 4.8: TOPICS database table.

Post Model

Each post has a body, a text field containing the body of the post, and a user ID of the user who created the post. Any logged-in user can create a post, only moderators (or OAs) can edit or delete posts. Table 4.9 shows the POST model's database structure.

Name	Type	Description
id	integer	The primary key
body	text	Body of the post
created_at	datetime	The date and time that the post was created
updated_at	datetime	The date and time that the post was last edited
topic_id	integer	ID of the topic that this post belongs to
user_id	integer	ID of the user who created this post

Table 4.9: POSTS database table.

Figure 4.9 shows the relationships among forums, topics, and posts.

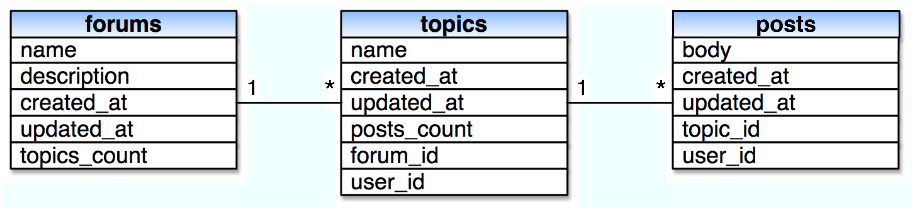


Figure 4.9: Entity relationship diagram for forums, topics, and posts.

Controllers

The Forum, Topic, and Post controllers are all standard REST-style controllers with the basic CRUD functions of a REST resource. Since each topic belongs to a particular forum, and each post belongs to a particular topic, these resources are nested. The topics resource is nested beneath a forum resource, assessable via URLs such as `/forums/1/topics` and `/forums/1/topics/1`. The posts resource is nested beneath a topic resource and, in turn, a forum resource. Therefore, the posts resource is assessable via URLs such as `/forums/1/topics/2/posts` and `/forums/1/topics/2/posts/2`.

4.6.5 User Blog with Web Services Support

A blogging service allows each Ontoverse user to create a number of blog entries and each entry can be commented by other members of the community. The blog structure consists of the two models Entry and Comment. A user's blog has many entries, and an entry many comments. The User model is extended to hold attributes common to an entire blog.

The blog service implements some features of established blog APIs, making it possible to use a desktop blogging client to add blog entries.

Entry Model

Each blog entry consists of a title and the body text and belongs to a user. A flag defines whether the post has been published or if it is just a draft. The creation and last-update time of the entry will also be stored. A counter cache keeps track of how many comments there are for each entry. The database fields necessary for the Entry model are shown in Table 4.10.

Comment Model

The Comment model holds the details of the comments left for each blog entry. Since only registered users of the site are allowed to leave comments, the ID of the user who

Name	Type	Description
id	integer	The primary key
title	string	Title of the blog entry
body	text	Body text of the blog entry
created_at	datetime	The date and time this entry was created
updated_at	datetime	The date and time this entry was last updated
comments_count	integer	The counter cache of the number of comments for this entry
user_id	integer	ID of the user to whom the entry belongs

Table 4.10: ENTRIES database table.

left the comment is stored, along with details of the comment including which entry the comment refers to, the body text, and the creation date and time. The database fields necessary for this model are shown in Table 4.11.

Name	Type	Description
id	integer	The primary key
body	text	Body text of the comment
created_at	datetime	The date and time this comment was created
entry_id	integer	ID of the entry that this comment belongs to
user_id	integer	ID of the user who created this comment

Table 4.11: COMMENTS database table.

User Model (extended)

The USER model is added by a number of fields to support the blogging features. Users can set titles for their blogs and enable or disable commenting on their blogs. A counter cache keeps track of the number of entries that a user has created in the blog. The additional fields required for the existing USER model are shown in Table 4.12.

Figure 4.10 shows the relationships among users, entries, and comments.

Name	Type	Description
blog_title	string	Title of the user's blog
enable_comments	boolean	Blog can be enabled (true) or disabled (false)
entries_count	integer	The counter cache of the number of entries created by this user

Table 4.12: USERS database table with additional fields (shortened).

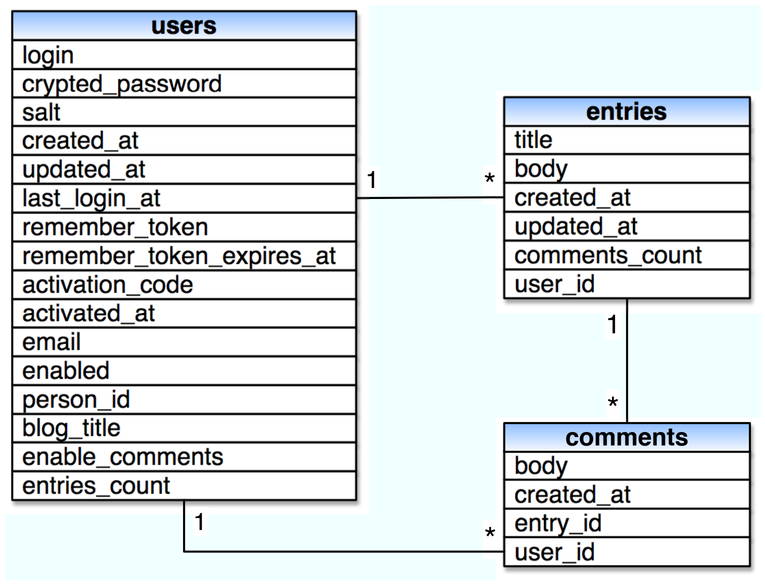


Figure 4.10: Entity relationship diagram for users, entries, and comments.

Controllers

Blogs Controller The BLOGS controller provides an entry portal to the blogs, listing the ten most recently updated blogs.

Entries Controller The ENTRIES controller provides access to the user's blog. Each collection of entries belongs to a specific user, and is nested within the users resource via URLs such as `/users/1/entries` and `/users/1/entries/2`. For these features, a standard REST-style controller is used. The `new`, `create`, `edit`, `update`, and `destroy` methods are only accessible by the owner of the blog, allowing that user to maintain and post to the personal blog. The `show` method displays a specific entry along with all comments left for that entry, and the `index` action provides the standard blog view.

Comments Controller This controller just requires a `create` method to actually save a new comment, and a `destroy` action to allow the owner of a blog to delete any comments if desired. The other five standard REST-style methods are either accessible by the ENTRIES controller or not required.

Blogging Interface

The Blogger API[6] provides some basic methods for blogging support as an XML-RPC web service shown in Table 4.13.

Method	Description
`blogger.getUsersBlogs(appkey, login, password)`	Returns information about the blogs one user have
`blogger.getUserInfo(appkey, login, password)`	Returns information about a specific user
`blogger.getPost(appkey, postid, login, password)`	Returns the content of a specific blog post
`blogger.getRecentPosts(appkey, blogid, login, password, number_of_posts)`	Returns a list of the most recent blogs posts in a particular blog
`blogger.newPost(appkey, blogid, login, password, content, publish)`	Creates a new post on a particular blog
`blogger.editPost(appkey, postid, login, password, content, publish)`	Changes the content of a blog post

Table 4.13: Blogging interface web service database table.

Backend Controller The BACKEND controller converts incoming method invocation requests into the API method calls (*dispatching*) and takes care of sending back the responses. Here layered dispatching is used, which allows to implement multiple APIs with one controller. The overview about layered dispatching is given in Figure 4.11.

[6]`http://code.google.com/apis/blogger/overview.html`

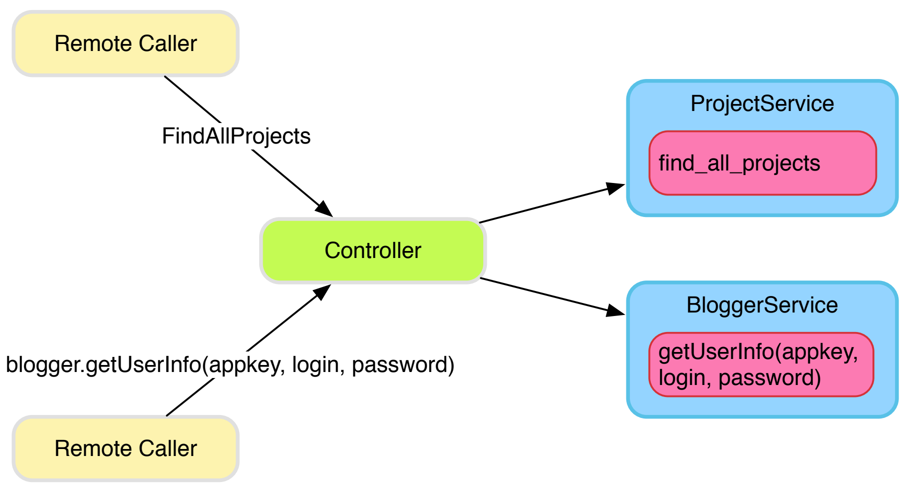


Figure 4.11: Overview of layered dispatching with two APIs and one controller (e. g. BACKEND controller).

4.6.6 User Photos

A photo gallery allows users to upload any photos to their gallery, where photos are displayed in a thumbnail view. This will encourage users to get involved on the ONTOVERSE platform and to make it more personal. Per default the latest uploaded photo from a user is displayed on the user's profile page as a thumbnailed version but it can be exchanged by another photo.

The user images are stored as files on the server filesystem instead of using a database, since filesystems are highly optimized to deliver static content such as images, while database connections are potentially expensive.

The photo gallery feature consists of only one PHOTO model. Each photo belongs to one user, and, in turn, a user has many photos.

Photo Model

Besides database fields required for photo upload, the PHOTO model consists of a title and a description. Table 4.14 shows the complete structure of the PHOTO model's database.

Name	Type	Description
id	integer	The primary key
title	string	The title of the photo
body	text	Description of the photo
created_at	datetime	The date and time the photo was uploaded
filename	string	The original file name of the uploaded photo
content_type	string	The MIME type of the uploaded photo
size	integer	Size of the uploaded photo
height	integer	Height of the photo in pixel
width	integer	Width of the photo in pixel
thumbnail	string	Name of the size of a thumbnail
parent_id	integer	ID of the thumbnails parent file

Table 4.14: PHOTOS database table.

Controllers

Photo Controller The PHOTO controller enables access to view all photos on the site at the root level. This resource only needs to implement an `index` action, since all the other actions need to be accessed via a nested resource.

UserPhotos Controller The USERPHOTOS controller allows to display the photos from a specific user. This controller provides the usual REST CRUD actions accessible as a nested resource below the users resource such as `/users/1/photos` and `/users/1/photos/1`. Only logged in users can upload photos or edit the attributes of existing photos.

4.6.7 E-mail Messages and Newsletter

In this section, the e-mail message sending functionality and the e-mail newsletter feature are described.

E-mail sending automatically directly to users is useful, such as sending a welcome mail when they sign up, offering them to receive news articles as e-mails or mail to allow them to reset their passwords. Furthermore, it can be used as an automated mailer that informs users when someone joins an ontology project or left a new comment in someone's blog. For example, this enables a PA or the blog's owner to quickly react to the message.

The newsletter feature allow OAs to easily create and send newsletters or notices to all users of the site. This could be necessary to notify users of upgrades or new features. It is made possible for users to turn e-mail notifications on or off if users do not want to have e-mail notifications or newsletters at all.

E-mail Messages

The e-mail should at least contain the e-mail address of recipients, the e-mail address that the mail is to be sent from, a subject for the message, and of course a body text. It can have both plain text and HTML parts, meaning that the mail displays only plain text on text-based e-mail applications and as an HTML e-mail in applications that support it. The HTML version of the same e-mail can have more interesting parts by adding links to the author's profile or the resource reference the e-mail originates from.

Newsletter

Newsletter Model The NEWSLETTER model stores the e-mail subject, body text, and the time and date that it was created. It also stores whether the newsletter has been sent or not and if so, the date and time it was sent. This database structure is shown in Table 4.15.

Name	**Type**	**Description**
id	integer	The primary key
subject	string	The e-mail subject line
body	text	Body of the e-mail
sent	boolean	Whether this newsletter has been sent or not
created_at	datetime	The date and time this newsletter was created

Table 4.15: NEWSLETTERS database table.

Newsletter Controller The NEWSLETTER controller permits OAs to create and edit newsletters, which can then be sent by clicking a button on the newsletter display page.

All outgoing e-mails are queued in this database table, freeing the application from having to wait for a remote SMTP server. The database table can be processed by a separate application running as a daemon, or added to a scheduler.

4.6.8 Friends Network

A friendship system is implemented, offering users to easily add other users to their friends list. This list can be viewed by any user, and furthermore, users can quickly see the latest activities of their friends — this is limited to showing only simple information about particular actions. This enables users to keep up to date with their friends' activities.

The friendships are enhanced by adding metadata based on the XHTML Friendship Network (XFN) microformat specification[7], to specify the different kinds of relationships one has with other users. Adding semantic information with microformats[8] allow users and applications to easily understand the relationships among other users and their friends.

Friends Resource

The Friends resource represents the unique friends list for each user. This resource is nested beneath the USERS controller, accessible through URLs like `/users/1/friends` and `/users/1/friends/2`. It provides the usual CRUD functions to view a list of all of one's friends, to display a friend page view, to allow to set friendships attributes according to the XFN specification, and to remove a friendship.

[7]`http://gmpg.org/xfn/`

[8]`http://microformats.org/`

User Model (extended)

The User model is further extended by two fields to store the latest activity performed by each user. An activity gets recorded by selectively choosing what to store and when to store it inside the different models. The additional fields required for the existing User model are shown in Table 4.16.

Name	**Type**	**Description**
last_activity	string	Description of the last activity performed by the user
last_activity_at	datetime	The date and time that this activity was performed

Table 4.16: Users database table with two additional fields (shortened).

Friendship Model

The Friendship join model stores the friendships and the information about the relationships among users in a separate database table (Figure 4.12). This database schema is shown in Table 4.17.

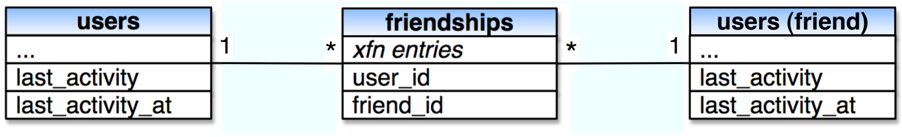


Figure 4.12: Friendship as a join model links together a user and a friend (both from User model).

Friends Controller

The Friends controller allows a user to create, modify, remove, and view friends. To add or edit new friendships, a user can select all XFN relationship values consisting of groups of mutually exclusive characteristics. The index view shows a user's list of friends, along with the last activities and links to their profiles. The own friend list view is completed by links to edit or remove each relationship. Every relationship in

Name	Type	Description
id	integer	The primary key
xfn_friend	boolean	Someone you are a friend to
xfn_acquaintance	boolean	Someone you have exchanged greetings with
xfn_contact	boolean	Someone you know how to get in touch with
xfn_met	boolean	Someone who you have actually met in person
xfn_coworker	boolean	Someone a person works with, or works at the same organization as
xfn_colleague	boolean	Someone in the same field of study/activity
xfn_coresident	boolean	Someone who lives at the same address as you
xfn_neighbor	boolean	Someone who lives nearby
xfn_child	boolean	A person's genetic offspring, or some that a person has adopted and takes care of
xfn_parent	boolean	One of your parents
xfn_sibling	boolean	A brother or sister
xfn_spouse	boolean	Someone you are married to
xfn_kin	boolean	Another relative
user_id	integer	ID of the creator and owner of this friendship
friend_id	integer	The user to whom this friendship refers

Table 4.17: FRIENDSHIPS database table based on XFN.

the friends list, represented by a number of XFN attributes, has its own relevant icon to highlight it.

4.6.9 Tagging and Searching

Tagging

Tags are keywords that are used to describe a particular object. Tagging is a very useful way of categorizing items that makes it easy for users to search and browse objects. *Tag clouds* display the most popular tags as the largest, so one can quickly see the most popular topics, and make a great starting point to allow people to discover objects on the site. Another usage for tags is to find related objects that share most of the same tags as the current object. Currently projects, photos, and publications are extended by tagging support.

Requirements

There are two ways to view the data. A user can view all tagged objects or only user-specific objects. To view tags by all users, a normal REST resource is created to retrieve tags from the corresponding model. To view tags that only belong to one user, the REST resource is nested beneath the users resource such as `/users/1/tags` and `/users/1/tags/anatomy`.

Relationships

Each model with tagging support is related to a Tag model through a third model called Tagging. There are two database tables required. The first, Tags table, stores the tag names and is shown in Table 4.18.

Name	Type	Description
id	integer	The primary key
name	string	Tag name

Table 4.18: Tags database table.

The Tagging table uses *polymorphic associations*, where a model is associated with objects of more than one model class. This database table is shown in Table 4.19. Figure 4.13 illustrates a polymorphic association.

Name	Type	Description
id	integer	The primary key
tag_id	integer	ID of the tag
taggable_id	integer	ID of the taggable object
taggable_type	integer	The model name of the taggable object
created_at	datetime	The date and time that this tagging was created

Table 4.19: Tagging database table.

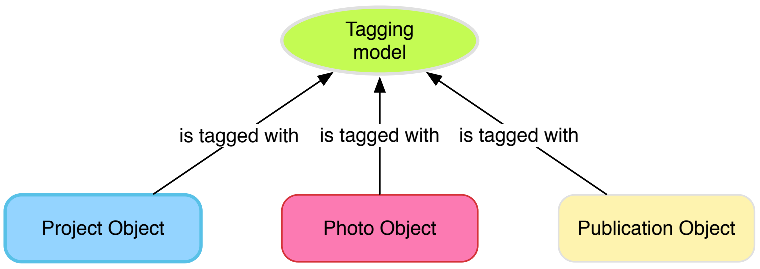


Figure 4.13: Tagging of projects, photos, and publications by means of polymorphic association.

Controllers

Tags Controller The TAGS controller is implemented to display all of the tags that have been added to taggable objects on the site, regardless of user. These tags are displayed as a tag cloud and can be limited to a smaller number of tags. Additionally only objects are displayed that match a particular tag.

User Tags Controller This controller acts in a similar way to TAGS controller, except that it only shows tags and taggable objects for a specific user.

Taggable Objects Controllers These controllers offer methods for users to add and remove tags from the corresponding taggable objects.

Searching

For performing searches on structured data, the kind of data in databases, a full-text engine called Ferret[9] is used when database loads get higher and there is more than one database table involved in the search. Currently searching includes user names and log-ins, project names and tagged objects.

[9]Ferret is a high-performance, full-featured text search engine library written for Ruby. It is inspired by Apache Lucene Java project (`http://lucene.apache.org/`).

4.6.10 Integrating other Web Applications

Many web applications offer public APIs with REST, XML-RPC, or SOAP interfaces to retrieve or save data on the site. Most Web 2.0 sites are moving toward REST architectures to offer very simple and lightweight interfaces to their data.

ONTOVERSE integrates with some other web applications, creating what has become known as a *mashup*, which is using parts of existing web applications to build something new. Google Maps is used to allow users to add physical location data to their addresses and their uploaded images.

Requirements

To store the geographical location, the PHOTO model needs extended by fields to store the location data and also a preferences string, which will be set if the user wants this location to be shown on the photo page. The required fields are shown in Table 4.20.

Name	**Type**	**Description**
geo_lat	decimal	The latitude of the photo's location
geo_long	decimal	The longitude of the photo's location
show_geo	boolean	A user-settable option to determine if the location data is displayed to others

Table 4.20: Required fields by the PHOTO model for mapping data.

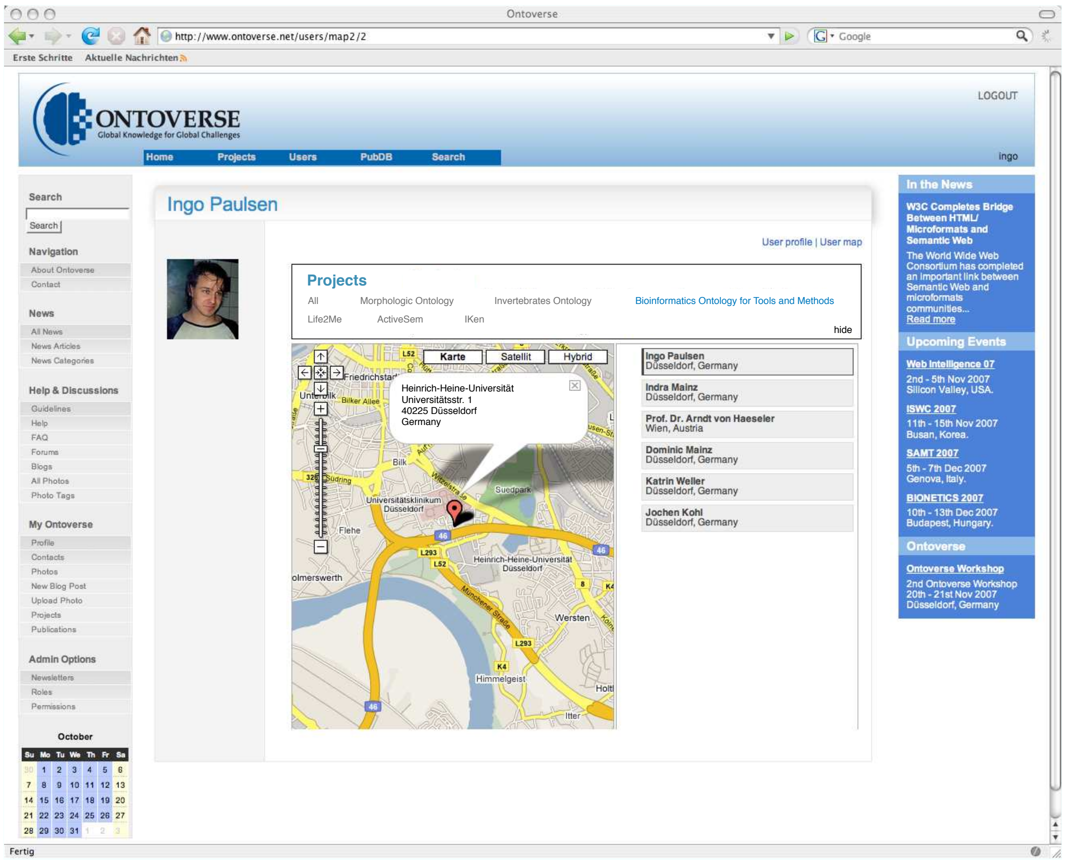
Mapping features

Getting locations There are three possibilities to obtain the location of an object:

1. Asking the user to manually enter latitude and longitude coordinates.
2. Using Google Maps Geocoder, which accepts an address and returns the coordinates of that point if traceable.
3. Allowing the user to drag and zoom in on the map to select a point on the map to set the coordinates.

The first and third possibilities are available for addresses and images, whereas the second one is only meaningful for address data.

Extended user's profile Mapping support allows a user to navigate through all user's addresses or his own ontology projects. A filtering mechanism selectively hides and shows groups of inhabitants or project members on the map. Figure 4.14 illustrates a user's profile together with his projects and the project's members depending on the selected project in the toolbar above.



Heinrich-Heine-Universität
Universitätsstr. 1
40225 Düsseldorf
Germany

Ingo Paulsen
Düsseldorf, Germany

Indra Mainz
Düsseldorf, Germany

Prof. Dr. Arndt von Haeseler
Wien, Austria

Dominic Mainz
Düsseldorf, Germany

Katrin Weller
Düsseldorf, Germany

Jochen Kohl
Düsseldorf, Germany

In the News

W3C Completes Bridge Between HTML/ Microformats and Semantic Web

The World Wide Web Consortium has completed an important link between Semantic Web and microformats communities...
Read more

Upcoming Events

Web Intelligence 07
2nd - 5th Nov 2007
Silicon Valley, USA.

ISWC 2007
11th - 15th Nov 2007
Busan, Korea.

SAMT 2007
5th - 7th Dec 2007
Genova, Italy.

BIONETICS 2007
10th - 13th Dec 2007
Budapest, Hungary.

Ontoverse

Ontoverse Workshop
2nd Ontoverse Workshop
20th - 21st Nov 2007
Düsseldorf, Germany

Figure 4.14: The map area is divided into three parts, one of which displays the location of the selected project member from the right side panel.

4.6.11 Ontology Projects

The building part and the connection to the other architectural parts of Ontoverse is an ontology project. Each project includes *one* ontology and is characterized by a name and description, information about its members, the creation data together with the name of the founder, and of course the ontology itself.

Every registered user is allowed to start an ontology project from scratch or to join an existing one with a given number of roles.

Project Model

In order to assign different projects to users, a Project model is created to store the different projects of users on the site. The required fields of the Projects table are shown in Table 4.21.

Name	**Type**	**Description**
id	integer	The primary key
name	string	Name of the project
description	text	Description of the project
ontology_name	string	The name of the project's ontology
subtitle	string	Subtitle of the project
abbreviation	string	Abbreviation or a short name for the project
created_at	datetime	The date and time that the project was created
updated_at	datetime	The date and time that the project was last updated
founder_id	integer	ID of the user who created the project

Table 4.21: Projects database table.

Project and User models are linked together using a join model Members. This model stores the IDs of the two separate models and links them together, and the date and time of creation and update. On the model site, a many-to-many relationship is specified by stating that a user has and belongs to many projects, and a project has and belongs to many users (Figure 4.15).

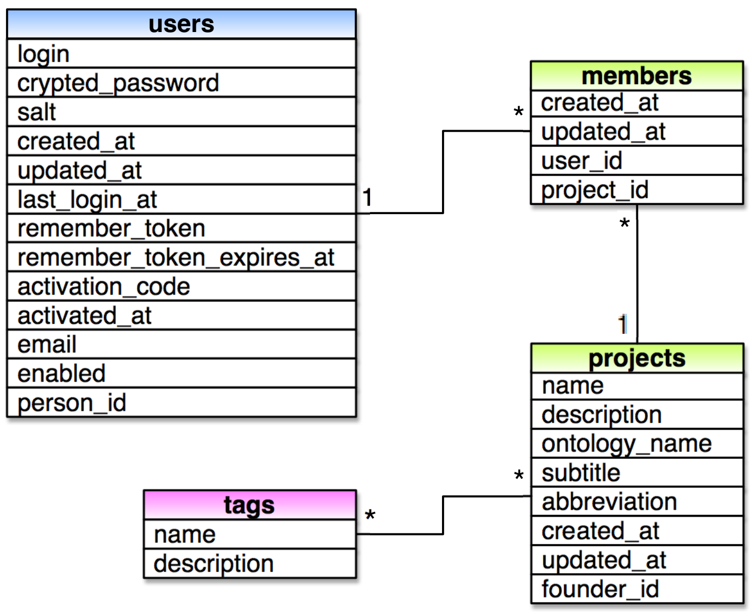


Figure 4.15: Entity relationship diagram for users, members, projects, and project tags.

Controllers

Project Controller The generated PROJECT controller is a CRUD controller that manipulates the resource PROJECT. This means that the controller belongs to exactly one resource type and offers a designated action for each of the four CRUD operations. (Additionally, the controller consists of the action `index`, to display a list of all resources of this type, the `new` action, for opening the new form, and the `edit` action, for opening the editing form.)

There are two ways to access ontology projects. An OA can access all projects with no restrictions, whereas project members can only view their own projects or other projects with limited access, i. e. mostly a general info profile. To view all projects, a normal REST resource is created to retrieve projects from the corresponding model. To view projects that only belong to one user or one member, the REST resource is nested beneath the users resource such as `/users/1/projects` and `/users/1/projects/bio2me`.

4.6.12 Project Wiki

Every ontology project has its own wiki page, which allows all project members to create, edit, and display project relevant articles through a web inferface.

Features

The project wiki offers the following features:

- A revision system follows changes on any article beginning from the first revision. It is easy to rollback to an earlier revision.
- Each revision is associated with an author, so it is possible to see who changed what.
- A *diff* algorithm tracks changes through revisions and lists the differences between two of them by means of variable colors.
- The article's content is markable supported by a WYSIWYG editor.
- Previewing allows to see exactly how the page will appear when saved.
- Connection to the site's general forum that can be used by members to discuss article-related topics and to provide feedback about the article's content.

Due to its design, this wiki is also suitable as a Content Management System (CMS) for OAs. As the ONTOVERSE site requires a number of information pages about the site's contents, as well as FAQ (Frequently Asked Questions) and help pages, this CMS supports to write and maintain pages, which do not change very often.

Internal structure

Each project page consists of a number of articles, which in turn, are dissected in many segments. A project page includes a wiki main page as kind of table of contents wherefrom all other article pages demerge. Per default, one article is set up for every project, comprising the ORSD of this ontology project. Figure 4.16 illustrates the structure of a project page.

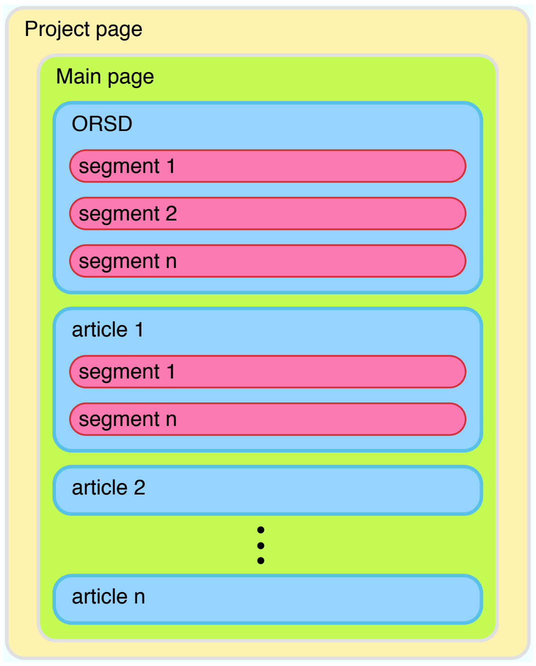


Figure 4.16: Internal structure of a project wiki page.

Relationships

The project wiki is represented by three models: ARTICLE, SEGMENT, and ATTACHMENT. A project has many articles, and an article has many segments and attachments. A single attachment can also be included in segments several times.

Article Model The ARTICLE model consists of a title and a content body. An article can be created and edited by all project members, deletion is only allowed for PAs. The ARTICLES database table is shown in Table 4.22.

An auxiliary table called CHANGES is joined with the PROJECTS and ARTICLES tables to keep track of article's changes. Essentially it logs the change modes and a summary about these changes.

Name	Type	Description
id	integer	The primary key
title	string	Title of the article
body	string	Content of the article
version	integer	The article's version number
created_at	datetime	The date and time that this article was created
updated_at	datetime	The date and time that this article was updated
project_id	integer	The project to which this article refers
creator_id	integer	ID of the creator of this article

Table 4.22: ARTICLES database table.

Segment Model Like an article, each segment has a title and a content body. Table 4.23 shows the SEGMENT model's database structure.

Name	Type	Description
id	integer	The primary key
title	string	Title of the segment
body	string	Content of the segment
version	integer	The segment's version number
created_at	datetime	The date and time that this segment was created
updated_at	datetime	The date and time that this segment was updated
creator_id	integer	ID of the creator of a particular segment's version
article_id	integer	ID of the segment's article

Table 4.23: SEGMENTS database table.

Attachment Model The ATTACHMENT model is similar to the PHOTO model in Section 4.6.6. Instead of storing a user's ID it stores a project ID, and a wiki ID. Figure 4.17 shows the relationships among projects, articles, segments and attachments.

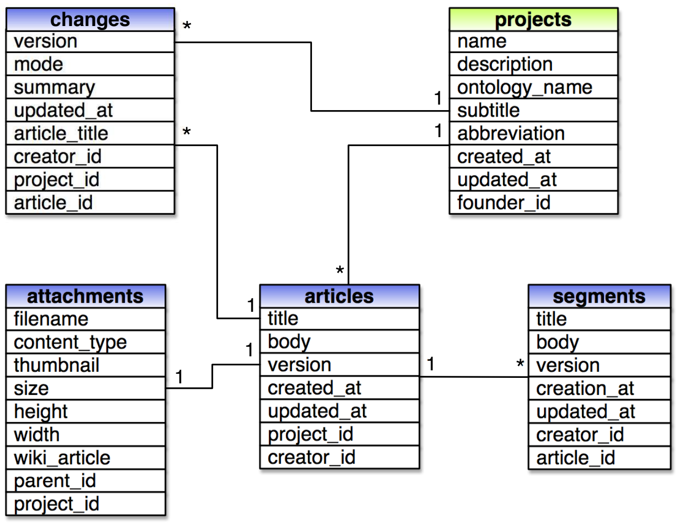


Figure 4.17: Entity relationship diagram for projects (with history of changes), articles, segments and attachments.

4.6.13 Publication Database: PubDB

This section gives an overview of the publication database in Ontoverse, called PubDB, to provide a document basis for building corpora with domain-relevant scientific publications. Each corpus is pre-annotated with XML tags and a prerequisite for IE:

Information Extraction The explosion of textual information requires new technologies that can recognize information originally structured for human consumption rather than for data processing. Information extraction is associated with template based extraction of information from language text, which was a popular task of the *Message Understanding Conferences* (MUCs) in the late eighties and nineties [108]. During the MUCs, there gradually arose a set of typical IE tasks [109, 110]. There are a number of typical IE tasks that lately have been extensively researched with regard to open domain IE. They include named entity recognition (NE), noun phrase coreference resolution (CO), template element construction (TE), template relation construction (TR) and scenario template production (ST).

NE is the recognition of names of people and organizations, place names, temporal expressions, certain types of numerical expressions, and terminology extraction. CO is the identification of (chains of) noun phrases that refer to the same object. It can be differentiated between nominal and pronominal co-reference. TE adds information (e. g. aliases, abbreviations, orthographical variants) to NE results using CO. TR is the identification of relations (e. g. taxonomic relations, property relations, and other static relations) between TE entities. ST aims at fitting the results of TE and TR into specified event scenarios, meaning that ST determines the dynamic relations between TEs.

PubMedLoader

PubMedLoader accepts search strings or keywords as input and searches for appropriate articles in PubMed[10]. At first only a list of PubMed IDs is returned not the whole

[10]PubMed is a very large corpus containing titles, abstracts, and other information about biomedical research articles. PubMed Central (PMC). `http://www.pubmedcentral.nih.gov`

documents. A user can select in this list all articles he wants, and after this procedure only those articles are fetched from PubMed and stored in PubDB. Figure 4.18 illustrates this searching and storing process with PubMedLoader.

PubDB Database model

The PubDB database model is organized in a *star schema*, which contains a single fact table (here Publications) plus a number of dimensional tables (Figure 4.19). These other tables represent additional information about a publication like its journal, keywords, and supplements with corresponding file types. All tables regarding the publication database model are described in the following.

Publications Publications is the central table in the PubDB database schema. It is connected to publication-specific tables but also to users, projects, people, and tags. Its attributes reflect individual metadata for publications such as title, author(s), abstract, journal, page numbers, and unique document identifiers. Other data resources in this table represent publication keywords and supplements (Table 4.24).

Journals The Journals table contains the journal title (and its abbreviation), NLM's unique journal identifier, the ISO abbreviation, and the print and electronic International Standard Serial Numbers (pISSNs and eISSNs) (Table 4.25).

Supplements and File Types The Supplements table stores supplemented material which provides additional information about a publication (Table 4.26). FileTypes table includes the associated file types (or formats) of the supplements (Table 4.27).

Keywords Keywords table stores all keywords and their general descriptions of all publications in PubDB (Table 4.28).

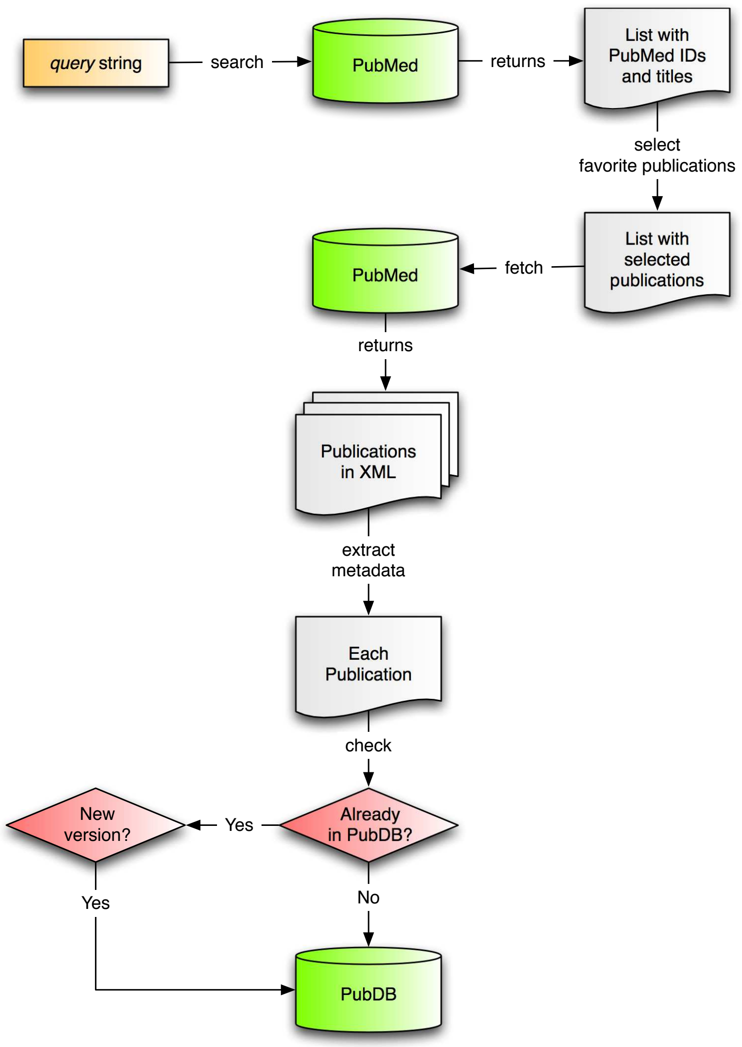


Figure 4.18: This flow chart illustrates the scenario where publications from PubMed are being stored into the ONTOVERSE publication database PubDB. At first a searcher expresses an information need using a formal statement called a *query*. The query is then given to PubMed to return a list of publications which satisfy the searcher's needs. Afterwards the searcher selects his favorite publications and fetches again to PubMed to return all publications in XML format. Each returned publication is extracted by metadata to check for availability in PubDB. New publications are stored in PubDB and existing publications in PubDB are updated by newer versions.

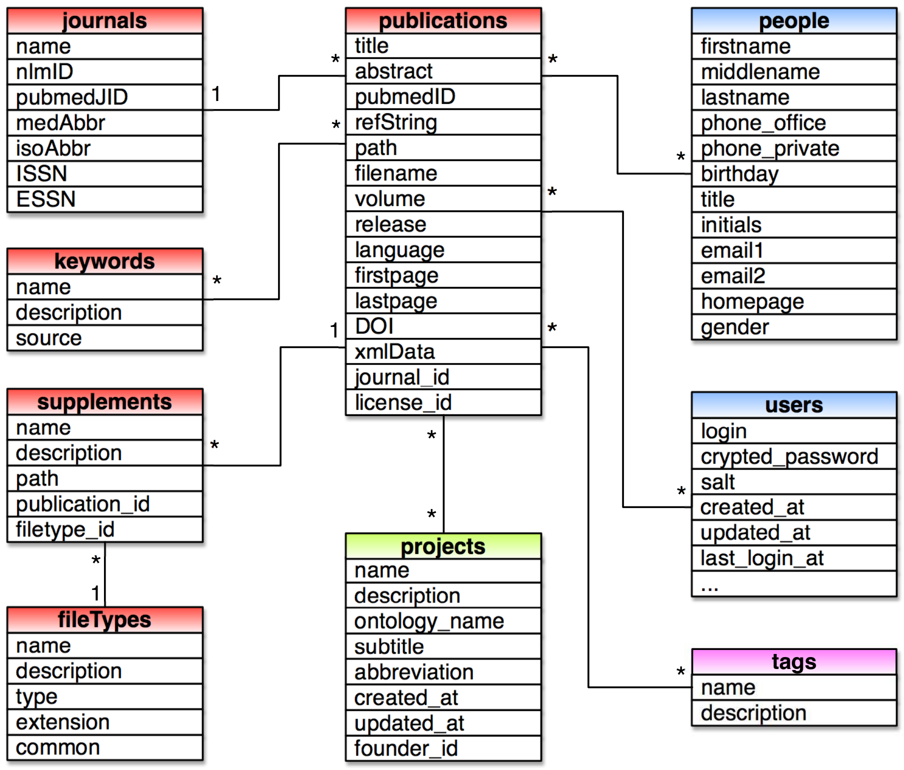


Figure 4.19: Entity relationship diagram for publications, users, people (authors), projects, and tagging (only relationships concerning publications are shown).

Name	Type	Description
id	integer	The primary key
title	string	The publication's title
abstract	text	The abtract of the publication
pubmedID	integer	PubMed identifier (PMID)
refString	string	Identification information
path	string	Path to publication pdf file
filename	string	Name of pdf publication file
volume	string	The number of the journal volume in which a publication is published
release	date	Publication release data
language	string	The language in which the publication was published
firstpage	integer	First page of the publication
lastpage	integer	Last page of the publication
DOI	string	Document Object Identifier
xmlData	text	Publication metadata in XML
journal_id	integer	Journal ID

Table 4.24: PUBLICATIONS database table.

Name	Type	Description
id	integer	The primary key
title	string	Title of the journal
abbr	string	Title abbreviation
NlmId	integer	ID assigned by National Library of Medicine (NLM)
pubmedJID	integer	PubMed Journal ID
MedAbbr	string	Standard abbreviation for the journal's title
ISOAbbr	string	ISO abbreviation
ISSN	string	International Standard Serial Number of the journal
ESSN	string	Electronic Standard Serial Number of the journal

Table 4.25: JOURNALS database table.

Name	Type	Description
id	integer	The primary key
name	string	The name of the supplement
description	text	Supplement description
path	string	Supplement path in the file system
publication_id	integer	Publication ID
filetype_id	integer	File type ID

Table 4.26: SUPPLEMENTS database table.

Name	**Type**	**Description**
id	integer	The primary key
name	string	The name of the file type
description	text	File type description
type	string	The file type
extension	string	File type extension
common	boolean	File format well-known or not

Table 4.27: FileTypes database table.

Name	**Type**	**Description**
id	integer	The primary key
name	string	The name of the keyword
description	text	Description of the keyword
source	string	Keyword's source

Table 4.28: Keywords database table.

Information Extraction Pipeline

Ontology Building Support The key idea behind IE in ONTOVERSE is to semi-automatically maintain ontologies by adding new instances (*ontology population*), as well as new concepts, properties and relations (*ontology enrichment*). This approach can keep the instances of the domain ontology up to date, by periodically re-training the IE system using a domain specific corpus.

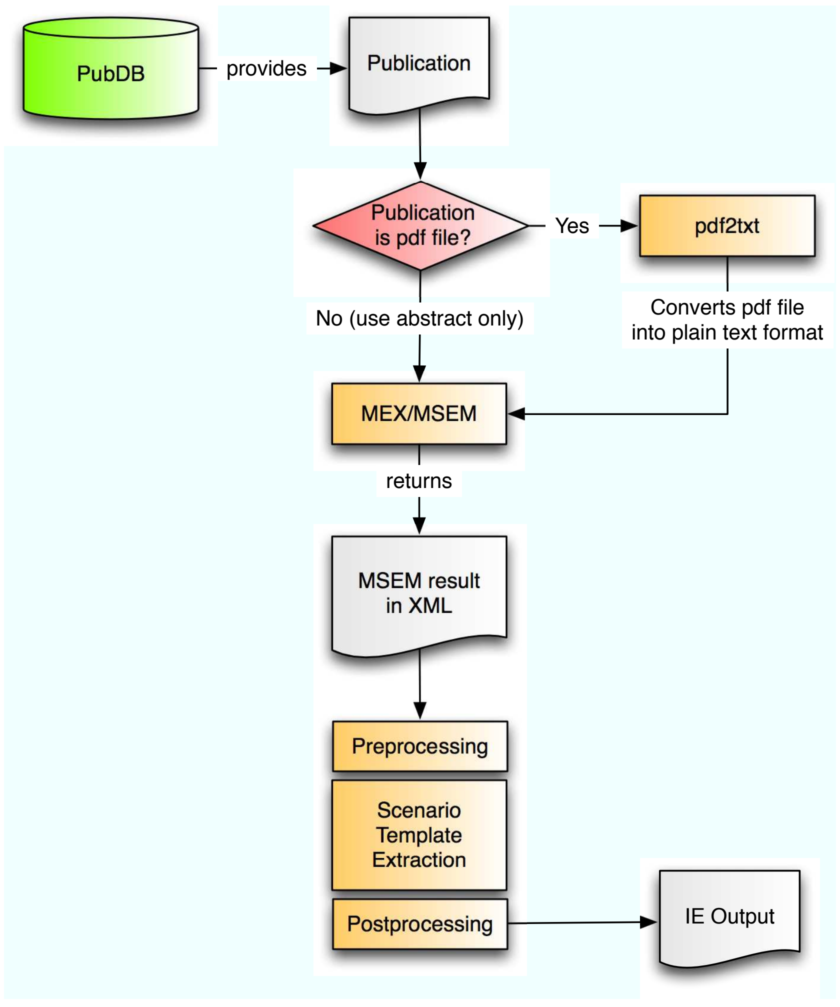


Figure 4.20: Information extraction pipeline in ONTOVERSE.

IE Pipeline Processing A project member selects a publication for IE. It is checked if this publication exists in pdf file format and if present it is converted into a plain text file. After this either the full text or only the abstract is used for the following IE tasks. Machinese Extractor (MEX)[11] extracts terms and recognizes named entities. All found entities are classified into sets of different types. MSEM semantically analyzes these sets by providing semantic role recognition as well as grammatical, lexical and sentential semantic features. The output file in XML is tagged with language specific information and hence prepared for the final *scenario template extraction* task (after an optional preprocessing step which definitely enumerates sentences if desired). After IE with several templates a postprocessing step might be useful to convert the extracted information in a special format (e. g. OWL or HTML with extracted regions colored). Figure 4.20 illustrates the IE as pipeline process.

[11] MEX and Machinese Semantics (MSEM) are constituents of the Connexor Machinese product family. `http://www.connexor.com/`

4.6.14 Collaboration Architecture

This section presents the middleware, and ontology editing and visualization concepts of the Ontoverse platform. This is a short summary of the German e-Science paper in [111]. These technologies described here do not represent the works of the author of this thesis, but they are presented here for completeness.

Back-end

The back-end persistently stores large amounts of data and grants concurrent access to it. Moreover it supports group and community awareness features, the management of different branches of an ontology, conflict resolution means during merge processes, and additional data like timestamps and other copyright information. The back-end is represented by a layer, called SWAT *Semantic Web Application Toolkit*, and consists of a blackboard architecture (SQLSpaces), several agents communicating with SQLSpaces, and a client as an interface between SQLSpaces and the web application (Figure 4.21).

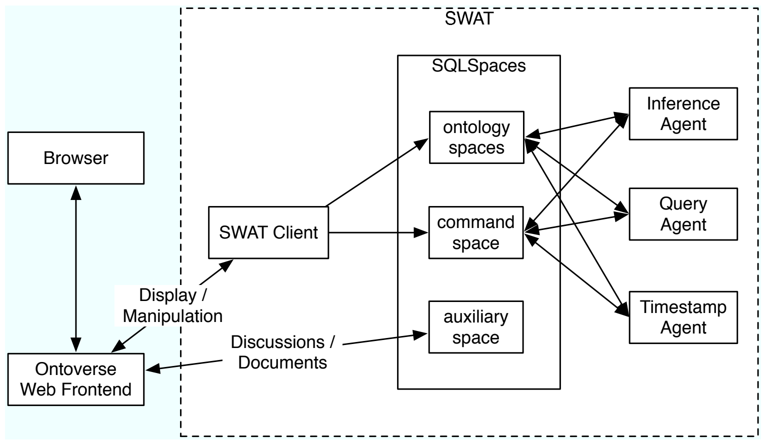


Figure 4.21: SWAT architecture.

SQLSpaces SQLSpaces is a flexible blackboard architecture with a virtual shared memory implementation called a *tuple space* (cf. [104]), which works on a relational database. A tuple space server provides access to several spaces, each representing an independent data storage. SWAT makes sure that each ontology lies in its own space, and are separated from messages exchanged by the agents.

Agents The agent architecture allows to implement several agents: An Inference Agent is written in Prolog and offers solutions for some problems related to ontologies like consistency checking. Another Prolog agent (Query Agent) translates tuple space operations into SQL statements. In order to guarantee copyright protection and advanced security features a Security Agent is capable of registering the data with timestamps.

Collaborative work

To support collaborative work on parts of ontologies two types of awareness modes are supported.

Asynchronous Collaboration This type is needed when several users work on the same ontology part, but not at the same time. A user check out a public version and thus creates a private workspace unseeable for other users. On the SQLSpaces layer this workspace is a branch of the public version that belongs to this specific user. When the user finishes his work he can commit this version to the original version or to a newer version. During the commit all differences between both versions are calculated and a list of conflicts is generated that could not be solved automatically (*optimistic locking* strategy). As soon as these conflicts are solved by the user the commit is executed and the result is available as a new public version.

Synchronous Collaboration The synchronous mode supports users to share a private workspace with other users, who will then also be able to modify the ontological data in this workspace. To prevent conflicts, ontological entities are locked during edits (*pessimistic locking* strategy). All changes are instantaneously visible for all users in the same workspace. All participants of the shared workspace get immediately notified about changes and write locks. If such a synchronous session is finished the private workspace can be committed and will become a new public version.

Ontology Editor

The ontology editor allows users to edit, visualize, compare and merge ontologies within an ontology project.

Editing offers a lightweight editor to change the structure of an ontology by adding, changing, and deleting concepts, properties, and descriptions. Ontologies can be compared and based on this comparison be merged into a new ontology.

Visualization consists of a graphical representation of an ontology as a touchgraph, which gives a user the big picture of the ontology and let him navigate, zoom in and out to discover clusters and interrelations of his interest. Furthermore the user interface offers a visualization called *SmartTree* for presenting the concept hierarchy in combination with graph views, which are especially useful to explore the network structure of large ontologies. Within the SmartTree, ontology concepts are presented as nodes and instances as special leaf types with different graphical representations.
The following features are implemented in SmartTree:

Focus and context A selected concept representation gets a bigger scale value than those concept representations in the distant areas inside the visualization. This focuses the concept of interest, whereas connections to the other concepts remain visual, so that the user has a better overview with respect to the concept hierarchy.

Property-Lines They connect the actual selected concept with concepts in the range of the OWL object property, so that the user gets an impression about the semantic information of the ontology.

Condense and Explode A flap mechanism for hiding subtrees in the concept hierarchy that the user is not interested in. The benefit for the user is to hide uninteresting parts of the concept hierarchy.

4.7 Usage Scenarios

This section describes four scenarios that deal with some functionalities of the ONTOVERSE platform in the context of social networking, ontology project organization, ontology population from extracted information and collaborative ontology editing.

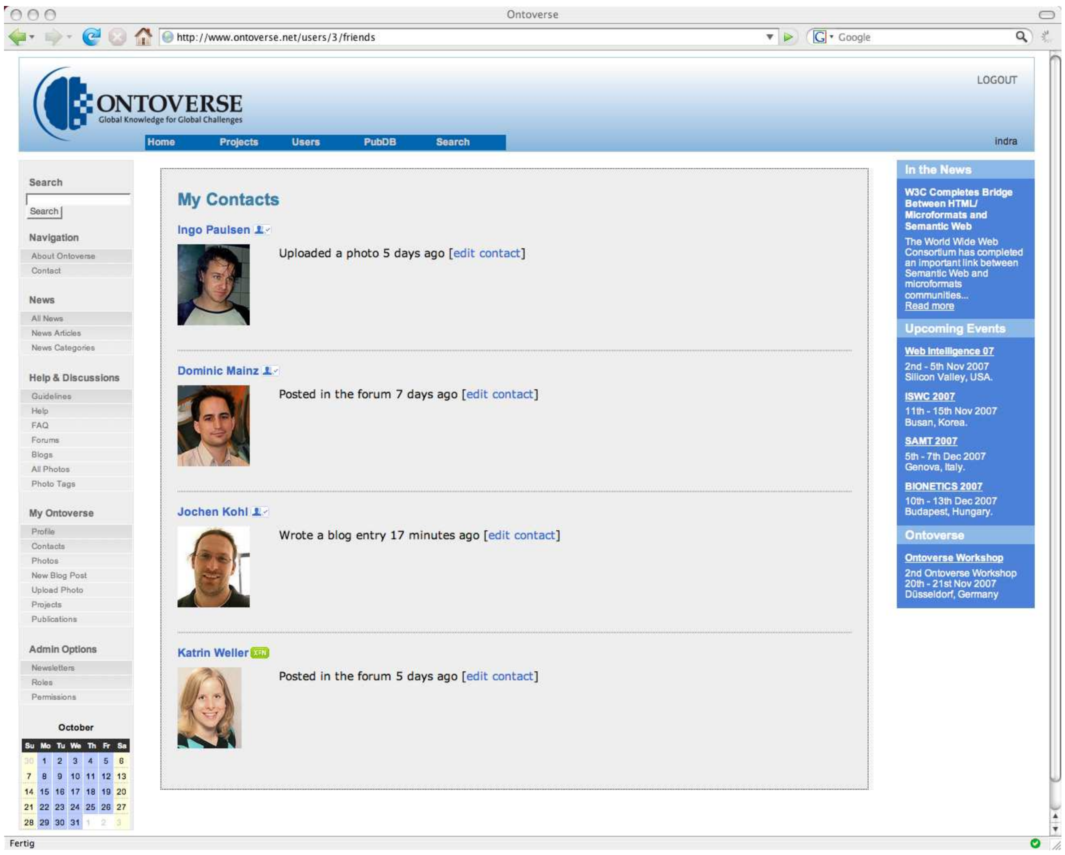
4.7.1 User Interaction/Networking

The registered user *Indra* invokes the ONTOVERSE site, logs in and is informed about the newest project members and ontology projects on the ONTOVERSE main page. Furthermore, she can gain insight about ONTOVERSE related news articles for all platform members, upcoming events of general interest and also events regarding only the projects she is associated with. In the following, one typical user activity is described in more detail:

1. *Indra* opens the 'Ontologies' forum page and selects topic 'Bio-Ontologies'. She reads some messages and clicks on *Katrin*, the author of one interesting response about her recently initiated ontology project called BIO2Me.
2. *Katrin's* user profile is shown.
3. A geographical map is selected to display *Katrin's* address. *Indra* can see that another member of her projects has the same address.
4. *Indra* decides to add *Katrin* as a contact and she has to approve *Indra's* request.
5. After confirmation *Katrin* is registered in *Indra's* contact list (Figure 4.22).

4.7.2 Project Organization

In this story *Katrin* asks the project administrator of BIO2Me *Indra* for permissions to join this project. After becoming a member she investigates the project wiki and wants to extend some wiki articles. These steps are listened in more detail:



Su	Mo	Tu	We	Th	Fr	Sa
30	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	1	2	3

My Contacts

Ingo Paulsen

Uploaded a photo 5 days ago [edit contact]

Dominic Mainz

Posted in the forum 7 days ago [edit contact]

Jochen Kohl

Wrote a blog entry 17 minutes ago [edit contact]

Katrin Weller XFN

Posted in the forum 5 days ago [edit contact]

In the News

W3C Completes Bridge Between HTML/ Microformats and Semantic Web

The World Wide Web Consortium has completed an important link between Semantic Web and microformats communities...
Read more

Upcoming Events

Web Intelligence 07
2nd - 5th Nov 2007
Silicon Valley, USA.

ISWC 2007
11th - 15th Nov 2007
Busan, Korea.

SAMT 2007
5th - 7th Dec 2007
Genova, Italy.

BIONETICS 2007
10th - 13th Dec 2007
Budapest, Hungary.

Ontoverse

Ontoverse Workshop
2nd Ontoverse Workshop
20th - 21st Nov 2007
Düsseldorf, Germany

Figure 4.22: Contact list showing all contacts, types of relationships with the XFN icons and the contacts' latest activities.

1. *Katrin* searches for 'BIO2Me' on the main page and selects the project with its complete name 'BioInformatics Ontology for Tools and Methods'.

2. The project's profile page is displayed with some statistics about memberships and also project tags.

3. She decides to join BIO2Me as a domain expert.

4. *Indra* receives *Katrin's* request, confirms it and registers her as domain expert.

5. *Katrin* enters the main wiki page, selects the ORSD of BIO2Me and updates the section about 'Domain experts'.

6. She browses the ontology classes (or concepts) and clicks on the wiki article ‘Class:Program’.

7. A new page with all superclasses and subclasses of ‘Program’ is opened. Additionally a concept graph (as attachment) is presented which shows relationships between ‘Program’ and other classes within BIO2Me (Figure 4.23).

8. *Katrin* is interested in the concept ‘StrAl’ and navigates to its article page by choosing this concept. After this the wiki article page ‘StrAl’ is opened and she extends it by some annotations.

4.7.3 Ontology Population

Indra enters the site. As a project administrator of BIO2Me she has all rights to edit this ontology in every respect. In this case she is interested in extending BIO2Me with the newest programs or tools regarding *structure alignments* from IE results. To achieve her goal the following steps are necessary:

1. *Indra* invokes the PubDB main page and enters the PubMed query string ‘structure alignments’ into the search text field.

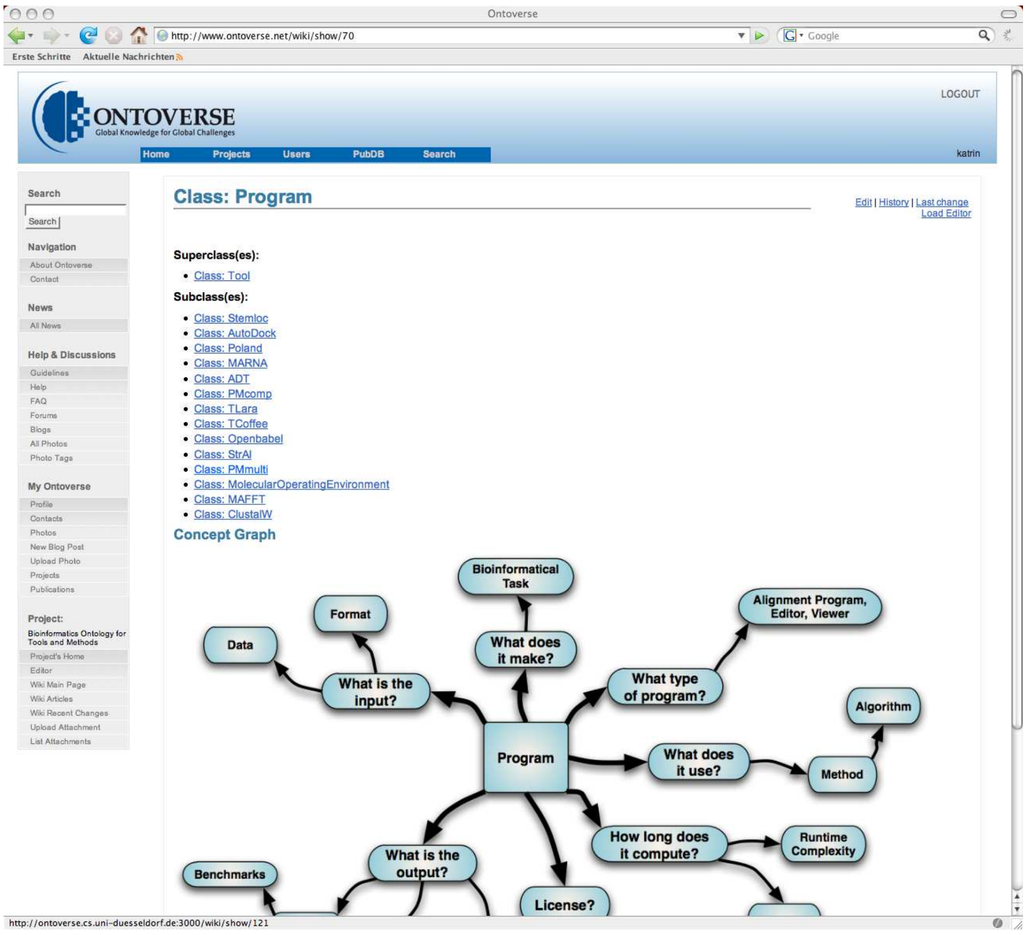
2. Two interesting results (‘MALIDUP’ and ‘StructSorter’) are selected and inserted into PubDB.

3. She decides to have both articles automatically tagged with BIO2Me’s concept entries.

4. After this both publications are also added to BIO2Me’s publication repository. She selects a collection of articles for information extraction (Figure 4.24).

5. The IE task is started with a special template to detect hierarchical relationships within publications.

6. She manually populates BIO2Me by meaningful extraction results via drag and drop (Figure 4.25).



Class: Program

Superclass(es):

- Class: Tool

Subclass(es):

- Class: Stemloc
- Class: AutoDock
- Class: Poland
- Class: MARNA
- Class: ADT
- Class: PMcomp
- Class: TLara
- Class: TCoffee
- Class: Openbabel
- Class: StrAl
- Class: PMmulti
- Class: MolecularOperatingEnvironment
- Class: MAFFT
- Class: ClustalW

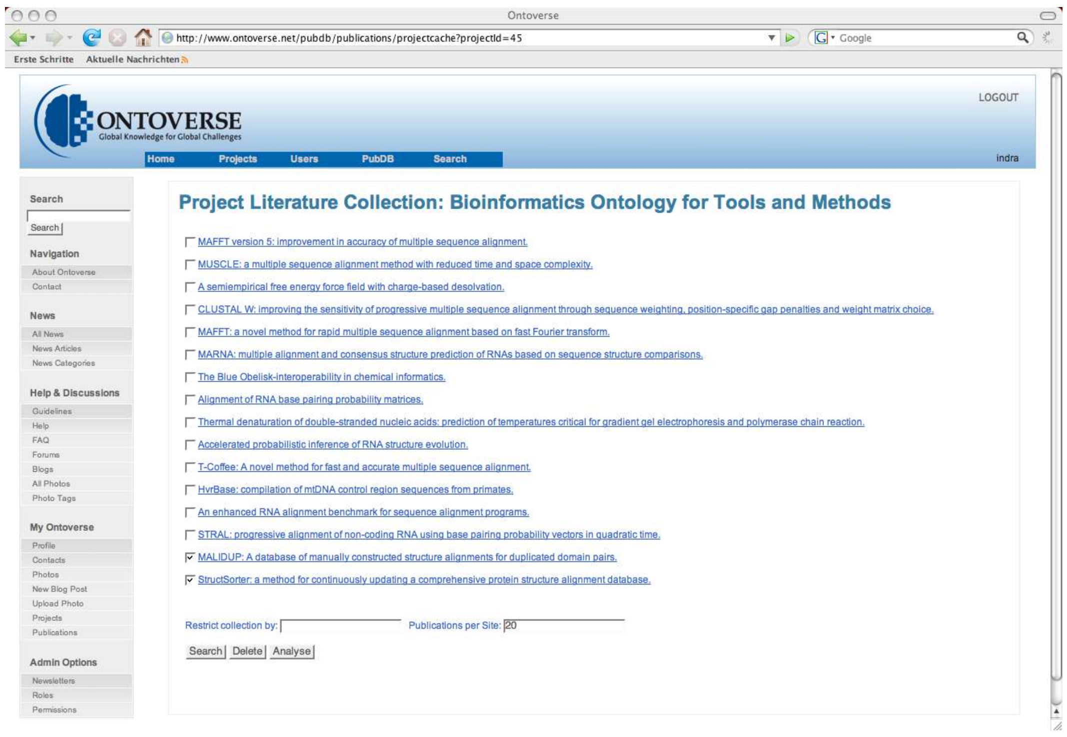
Concept Graph

Figure 4.23: BIO2Me's concept 'Program' wiki article page. It shows all superclasses and subclasses of a program and a concept graph as an attachment of this project wiki. Class articles are automatically generated from the ontology's classes.

7. BIO2Me's wiki page is updated with the new entries and *Indra* writes a message about one doubtable result up for discussion. (In this case she is not really sure if 'StructSorter' is an instance of subclass 'StructureAlignmentProgram'.)

4.7.4 Ontology Editing

Indra opens the ontology editor and explores the new structure alignment programs in the graphical representation of BIO2Me (Figure 4.26). In the meantime the BIO2Me member *Ingo* has also opened the ontology editor and adds the protein database Inv-



Project Literature Collection: Bioinformatics Ontology for Tools and Methods

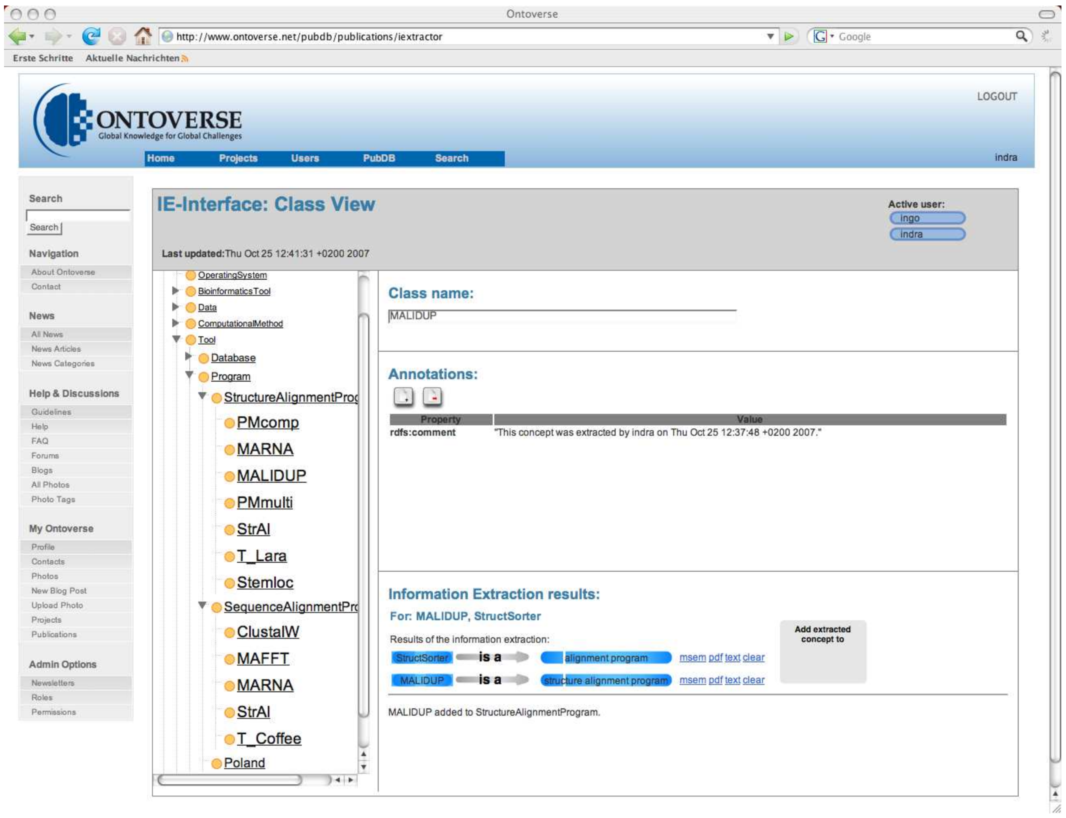
- ☐ MAFFT version 5: improvement in accuracy of multiple sequence alignment.
- ☐ MUSCLE: a multiple sequence alignment method with reduced time and space complexity.
- ☐ A semiempirical free energy force field with charge-based desolvation.
- ☐ CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice.
- ☐ MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform.
- ☐ MARNA: multiple alignment and consensus structure prediction of RNAs based on sequence structure comparisons.
- ☐ The Blue Obelisk-interoperability in chemical informatics.
- ☐ Alignment of RNA base pairing probability matrices.
- ☐ Thermal denaturation of double-stranded nucleic acids: prediction of temperatures critical for gradient gel electrophoresis and polymerase chain reaction.
- ☐ Accelerated probabilistic inference of RNA structure evolution.
- ☐ T-Coffee: A novel method for fast and accurate multiple sequence alignment.
- ☐ HvrBase: compilation of mtDNA control region sequences from primates.
- ☐ An enhanced RNA alignment benchmark for sequence alignment programs.
- ☐ STRAL: progressive alignment of non-coding RNA using base pairing probability vectors in quadratic time.
- ☑ MALIDUP: A database of manually constructed structure alignments for duplicated domain pairs.
- ☑ StructSorter: a method for continuously updating a comprehensive protein structure alignment database.

Restrict collection by: Publications per Site: 20

Search | Delete | Analyse

Figure 4.24: Project publication collection for the BIO2Me ontology. Two publications are selected for an upcoming information extraction analysis.

hogen as an instance of subclass 'DatabaseProtein'. During the editing process he gets notified that some other parts of the ontology have changed. He decides to compare both current ontology versions to see differences between them (Figure 4.27). As there are different spellings for some structure alignment programs that may arouse naming conflicts, BIO2Me's project members have to agree about unique naming notations to guarantee a successful ontology merging.



IE-Interface: Class View

Last updated: Thu Oct 25 12:41:31 +0200 2007

Class name:

MALIDUP

Annotations:

Property	Value
rdfs:comment	"This concept was extracted by indra on Thu Oct 25 12:37:48 +0200 2007."

Information Extraction results:

For: MALIDUP, StructSorter

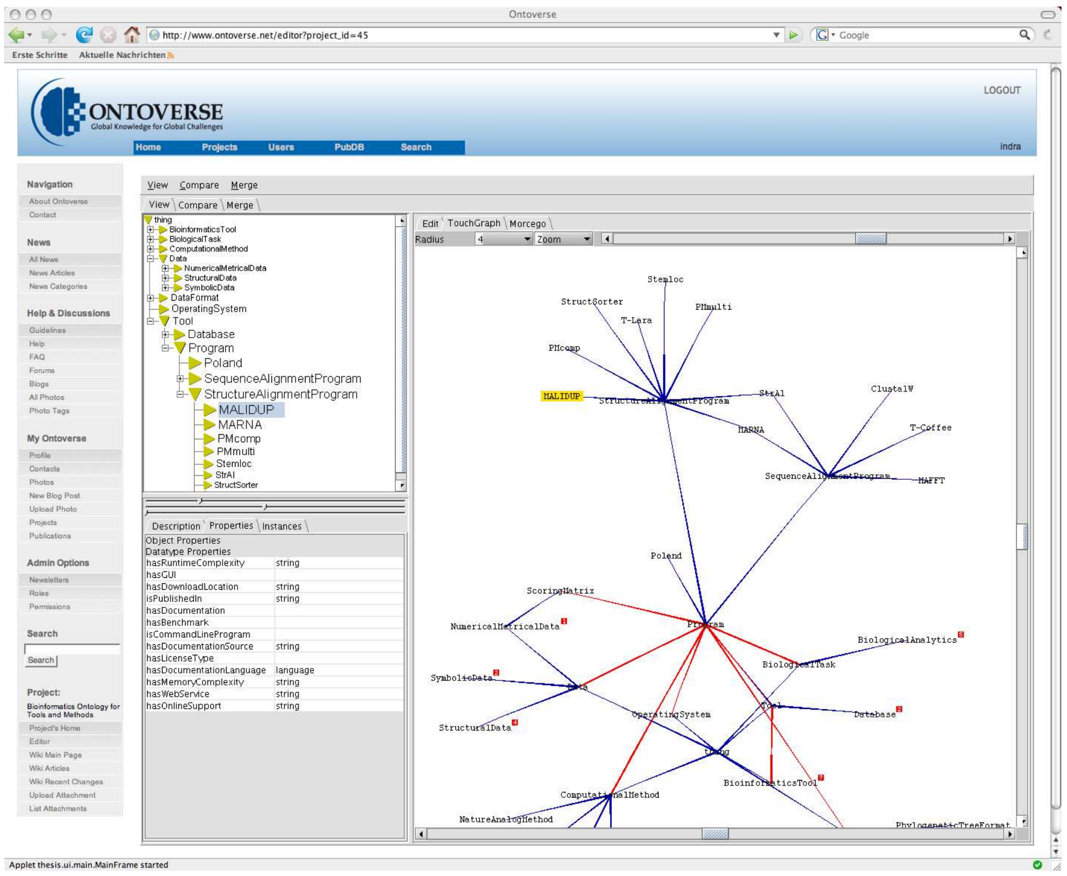
Results of the information extraction:

StructSorter is a alignment program

MALIDUP is a structure alignment program

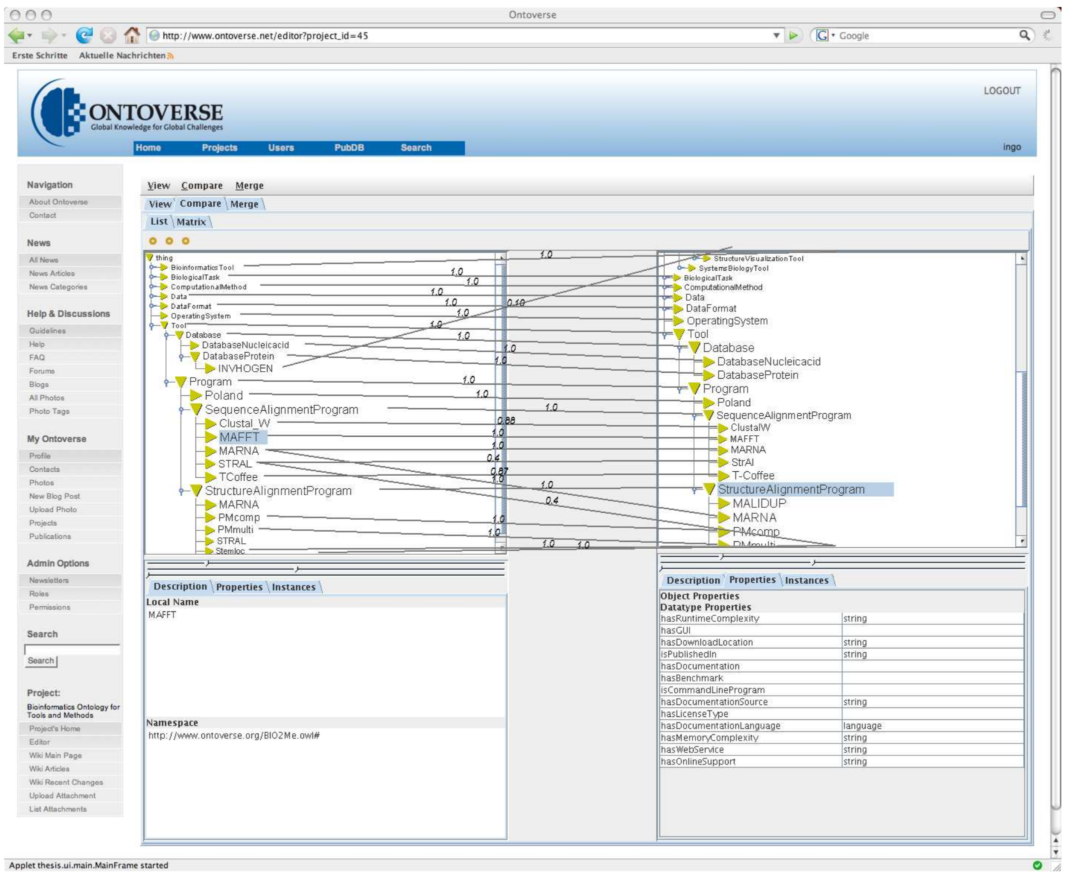
MALIDUP added to StructureAlignmentProgram.

Figure 4.25: Information extraction results are presented in the lower right part of the IE-Interface window. It shows that 'MALIDUP' and 'StructSorter' are alignment programs. Furthermore, 'MALIDUP' is qualified as structure alignment program and therefore classified into BIO2Me's class hierarchy via drag and drop (tree representation on the left side). For the present 'StructSorter' is also ranked as structure alignment program (Figure 4.26).



Object Properties	
Datatype Properties	
hasRuntimeComplexity	string
hasGUI	
hasDownloadLocation	string
isPublishedIn	string
hasDocumentation	
hasBenchmark	
isCommandLineProgram	
hasDocumentationSource	string
hasLicenseType	
hasDocumentationLanguage	language
hasMemoryComplexity	string
hasWebService	string
hasOnlineSupport	string

Figure 4.26: On the left the ontology editor illustrates the class view and the properties of the selected instance 'MALIDUP' at the bottom. 'MALIDUP' is highlighted in the graphical view on the right side. This view shows all aligment programs in BIO2Me in the top half and other classes below them.



```
thing
  BioinformaticsTool
  BiologicalTask
  ComputationalMethod
  Data
  DataFormat
  OperatingSystem
  Tool
    Database
      DatabaseNucleicacid
      DatabaseProtein
        INVHOGEN
    Program
      Poland
      SequenceAlignmentProgram
        Clustal_W
        MAFFT
        MARNA
        STRAL
        TCoffee
      StructureAlignmentProgram
        MARNA
        PMcomp
        PMmulti
        STRAL
        Stemloc
```

Local Name
MAFFT

Namespace
http://www.ontoverse.org/BIO2Me.owl#

```
      StructureVisualizationTool
    SystemsBiologyTool
  BiologicalTask
  ComputationalMethod
  Data
  DataFormat
  OperatingSystem
  Tool
    Database
      DatabaseNucleicacid
      DatabaseProtein
    Program
      Poland
      SequenceAlignmentProgram
        ClustalW
        MAFFT
        MARNA
        StrAl
        T-Coffee
      StructureAlignmentProgram
        MALIDUP
        MARNA
        PMcomp
        PMmulti
```

Object Properties
Datatype Properties

hasRuntimeComplexity	string
hasGUI	
hasDownloadLocation	string
isPublishedIn	string
hasDocumentation	
hasBenchmark	
isCommandLineProgram	
hasDocumentationSource	string
hasLicenseType	
hasDocumentationLanguage	language
hasMemoryComplexity	string
hasWebService	string
hasOnlineSupport	string

Figure 4.27: String based comparison of two BIO2Me versions. Some classes have different namings, e. g. STRAL and StrAl have only a value of 0.4 caused by upper and lower cases. However, they represent the same instance.

Chapter 5

Conclusion and Outlook

The necessity for ontology engineering, annotating, and integrating is uncontested. Ontologies are the core element for an upcoming Semantic Web. Furthermore, the so-called Web 2.0 initiatives aim at interconnecting communities on the web and enabling fruitful collaboration for private and business use as well as for scientific work in various research areas. Combining Semantic technology and Web 2.0 aspects holds potential for new methods in (scientific) knowledge management, communication and collaboration in research and development. In this context several applications are developed which deal with problems of knowledge processing, knowledge management and knowledge transfer.

This thesis also contributes to these fields of research focused on the life sciences community. Especially, bioinformatics is characterized by a high degree of cooperation among the researchers who contribute their part to the whole knowledge base of genomics and proteomics. It results within the scope of the Ontoverse research project with the objective to develop an Internet-based application for collaborative ontology engineering in terms of an ontology wiki.

A motivation at the beginning of this thesis for the Ontoverse project was the lack of high-value ontological annotations in sequence database entries. It might be eligible that prospective annotations — particularly if implemented collaboratively — reflect ontological representations more accurately not just to classify genes by sequence data but also by semantic information. Semantic annotation of sequence data as a new approach for knowledge production and data preparation is well supported by this cooperative platform.

Against this background the main aspect in this thesis is the documentation of the developed ontology wiki as integral part of the ONTOVERSE platform. This ontology wiki as a Web 2.0 architecture manages the scientific user community and is interconnected with the collaboration architecture consisting of the middleware and the visual ontology editor.

To assist the community in populating ontologies, ONTOVERSE integrates information extraction technologies that can propose new concepts and instances extracted from scientific publications. IE screens textual data to fill predefined templates with facts that can be used to extend knowledge bases after being curated by ontology designers. The publication database PubDB includes project-specific text corpora. In return, the newly developed ontologies themselves will help to retrieve relevant documents from the database.

In the first part of this thesis the author developed a database of homologous invertebrate gene families and a graphical application to allow one to simultaneously handle all the data available in INVHOGEN to analyze homology relationships: taxonomic information, sequence annotations, multiple sequence alignments and phylogenetic trees. In the second part the field of responsibility within the ONTOVERSE research project was the implementation of the ontology wiki to manage user communities and ontology projects, to maintain a publication database providing project-specific document collections and enabling an interface for an information extraction application. IE results are integrated into respective ontologies and added to the underlying ontology backend system.

ONTOVERSE is an ongoing project that started in October 2005. The features described in this thesis are essential for a collaborative ontology development framework, but we are working to extend and to improve ONTOVERSE. In particular, further work needs to be spent on sophisticated awareness mechanisms in the user interface to make the ODs and DEs aware of other people working in the same field. Furthermore, the integration of information extraction results into ontologies should be done semi-automatically.

With BIO2Me we present the fundament of a structured knowledge base in the field of bioinformatics tools and methods. The ontology so far covers the structure for the classification of bioinformatics programs. It features relevant information about the programs and therefore helps scientists to find adequate tools for their individual purposes. We intend the extension of BIO2Me with contributions from a larger number of experts on specific domains of the bioinformatics field, as building ontologies on such a substantial domain demands the knowledge of many researchers. To enable sophisticated queries in BIO2Me, we plan to develop an expert system operating on the basis of BIO2Me. The expert system should find details of entered programs as well as programs, that meet certain criteria.

In order to achieve the common goal, Ontoverse is a platform for collaborators to work and share perspectives, to view common work, and to interactively evaluate and critique each others' contributions.

Chapter 6

Fazit und Ausblick

Die Notwendigkeit zur Entwicklung, Annotation und Integration von Ontologien als Kernstück des Semantic Web ist unbestritten. Web 2.0 Technologien ermöglichen zudem die Vernetzung von Nutzergemeinschaften im Netz und stellen diesen Plattformen für die Zusammenarbeit bereit, sowohl für private als auch geschäftliche Zwecke und wissenschaftliche Arbeiten in verschiedenen Forschungsbereichen. Die Kombination semantischer Technologien mit Web 2.0 Ansätzen birgt Potential für neue Formen des (wissenschaftlichen) Wissensmanagements, der Forschungszusammenarbeit und Kommunikation. In diesem Zusammenhang werden aktuell verschiedene Anwendungen entwickelt, welche Probleme der Wissensaufbereitung, des Wissensmanagements und des Wissenstransfers für Forschung und Entwicklung aufgreifen.

Vor diesem Hintergrund ist auch der Beitrag dieser Arbeit zu sehen, welche den Einsatz von semantischen und kollaborativen Technologien speziell im Bereich der Life Sciences zum Thema hat. Dies ist besonders für die Bioinformatik relevant. Diese zeichnet sich durch ein hohes Maß an Kooperation zwischen Wissenschaftlern, beispielsweise in den Bereichen Genomik und Proteomik, aus. Die Arbeit entstand im Rahmen des ONTOVERSE Forschungsprojektes, dessen Ziel der Aufbau einer Internet-Plattform zur kollaborativen Erstellung von Ontologien ist.

Ausgangspunkt dieser Arbeit war der Mangel an hochwertigen Annotationen in Sequenzdatenbankeinträgen. In gemeinsamer Zusammenarbeit soll die Annotationsqualität durch Ontologieeinträge erhöht werden, damit Gene nicht nur anhand von Sequenzen, sondern auch mit semantischen Informationen klassifiziert werden. Zur Umsetzung dieser Ideen wird eine Kooperations-Plattform zur Erstellung fachspezifischer Ontologien benötigt.

Schwerpunkt dieser Arbeit ist die Implementierung einer solchen Plattform als grundlegender Teil des Ontoverse Projektes. Ein sog. Ontologie-Wiki nutzt neueste Web 2.0 Technologien zur Verwaltung der Benutzer-Community. Weiterhin dient es als Bindeglied sowohl zu der Kollaborations-Middleware (Backend) als auch zum graphischen Ontologieeditor.

Ontoverse integriert Technologien der Informationsextraktion und unterstützt so die Community bei der Erweiterung von Ontologien um neue Konzepte und Instanzen, die aus wissenschaftlichen Publikationen gewonnen werden.

Im ersten Teil der Arbeit wurde eine Datenbank für homologe Genfamilien von Invertebraten entwickelt (Invhogen). Eine graphische Benutzeroberfläche erlaubt die Betrachtung aller relevanten Daten in Invhogen bei der Analyse homologer Beziehungen: Informationen zu Taxonomien, Sequenzannotationen, multiple Sequenzalignments und phylogenetische Bäume. Im zweiten Teil der Arbeit wurde ein Ontologie-Wiki für die Benutzer- und Projektverwaltung entwickelt. Zusätzlich wurde eine Publikationsdatenbank implementiert, die Textkorpora aus wissenschaftlichen Publikationen verwaltet. Für die Integration von passenden Informationsextraktionsergebnissen in Ontologien wurde eine Schnittstelle zur Connexor Software programmiert und die Kommunikation mit dem Ontologie Backend zur Mitteilung der Ergebnisse gewährleistet.

Ontoverse ist ein laufendes Projekt, das im Oktober 2005 begonnen wurde. Die Entwicklungen, die in dieser Arbeit beschrieben wurden, sind grundlegende Bestandteile der Ontoverse Ontologieentwicklungsplattform, welche derzeit noch weiterentwickelt und verbessert wird. Es wird besonders daran gearbeitet die Benutzerschnittstellen und Kommunikationsformen zwischen Ontologiedesignern und Fachexperten zu erweitern und das Auffinden potentieller Ontologieprojektmitglieder zu erleichtern. Weiterhin soll die Integration der Informationsextraktionsergebnisse in Ontologien nicht nur manuell, sondern auch semiautomatisch erfolgen.

Mit BIO2Me wurde die Grundlage für eine Wissensbasis im Bereich bioinformatischer Werkzeuge und Methoden geschaffen. Bislang bildet diese Ontologie hauptsächlich die Struktur zur Klassifikation bioinformatischer Programme ab. BIO2Me bietet hierbei relevante Informationen zu einzelnen Programmen und unterstützt Wissenschaftler auf der Suche nach Programmen für ihre individuellen Verwendungszwecke. Die von uns geplante Erweiterung der BIO2Me Ontologie bedarf der Mithilfe weiterer Fachexperten aus verschiedenen Wissensbereichen, um die Domäne bioinformatischer Werkzeuge und Methoden noch weiter reichend abzudecken. Zu einem späteren Zeitpunkt wird mit der Erstellung eines Expertensystems begonnen, das den Zugriff auf die in der Ontologie gespeicherten Informationen erleichtern wird.

Ontoverse dient als Plattform, um Wissenschaftler untereinander zu vernetzen und bei der Bearbeitung von Forschungsfragen zu unterstützen. Wissen wird somit im gemeinsamen Dialog interaktiv ausgearbeitet und bereitgestellt.

Appendix A

Table & Database Schema

A.1 INVHOGEN

A.1.1 Attributes Assignments of a Gene Family

Table A.1 shows the assignment of attribute values of the selected GF `INV000805` from INVHOGEN, and one of its six GF entries with the accession number `P82706`.

A.2 Ontoverse

A.2.1 Ontology Wiki Database Schema

Figure A.1 gives an overview of the ONTOVERSE ontology wiki database schema. The types in all second columns represent migration types in Rails which are mapped to individual database adapters, e. g. MySQL and Oracle. For instance, a column entry declared to be `:integer` in a migration would have the underlying type `int(11)` in MySQL and `number(38)` in Oracle.

	Attribute	Value
GeneFamily	alignment	IM03_DROME MKFLSLA--FVLGLLALANATPLNP--GNVIINGDCRVCNVRA-- Q9V8G2_DROME MKWMSLV--FLCGLLAMAVASPLNP--GNVIINGDCRHCNVRGG- Q8IME0_DROME MKLLSIT--FLFGLLALASANPLSP--GNVIINGDCKVCNIRGD- IM01_DROME MKFFSVVTVFVLGLLAVANAVPLSPDPGNVIINGDCRVCNVHGGK IM02_DROME MKFFSVVTVFVFGLLALANAVPLSPDPGNVVINGDCKYCNVHGGK Q9V8F7_DROME MRFFAIVTVFVLGLLALANAIPLSPDPGNVIINGDCVNCNVRGGK *: :::. *: ****:* * **.* ***:***** **::.
	gfDescription	Immune-induced peptide 1,2,3 precursor
	gfIdentifier	INV000805
	numberOfSequences	6
	numberOfSpecies	1
	phylogeneticTree	(IM01_DROME:0.024937,IM02_DROME:0.097221,(Q9V8F7_DROME:0.122833,((IM03_DROME:0.060520,Q9V8G2_DROME:0.258919):0.055613,Q8IME0_DROME:0.256255):0.055475)):0.050067);
	taxonomy	Drosophila melanogaster Arthropoda Brachycera Diptera Drosophila Drosophilidae Endopterygota Ephydroidea Hexapoda Insecta Muscomorpha Neoptera Pterygota
GeneFamilyEntry	accessionNumber	P82706
	entryDescription	Immune-induced peptide 1 precursor (DIM-1)
	entryName	IM01_DROME
	taxID	7227
	wholeEntry	ID IM01_DROME STANDARD; PRT; 45 AA. AC P82706; Q9V8F6; ... //
GOTerm	identifier	GO:0005576
	minDistance	5
	maxDistance	5
	name	defense response
	quantity	2
	subOntology	Biological Process (P)
Taxonomy	taxID	7227
	parentIdentifier	32351
	rank	species
	scientificName	Drosophila melanogaster
	genbankCommonName	fruit fly
	misspelling	Drosophila melangaster
Sequence	type	Protein
	length	45
	molecularWeight	4670
	sequence	MKFFSVVTVFVLGLLAVANAVPLSPDPGNVIINGDCRVCNVHGGK

Table A.1: Attribute values in GeneFamily, GeneFamilyEntry, GeneOntologyTerm (GOTerm), Taxonomy, Sequence tables for GF INV000805, GO term GO:0005576, and one of six GF entries (with accession number P82706).

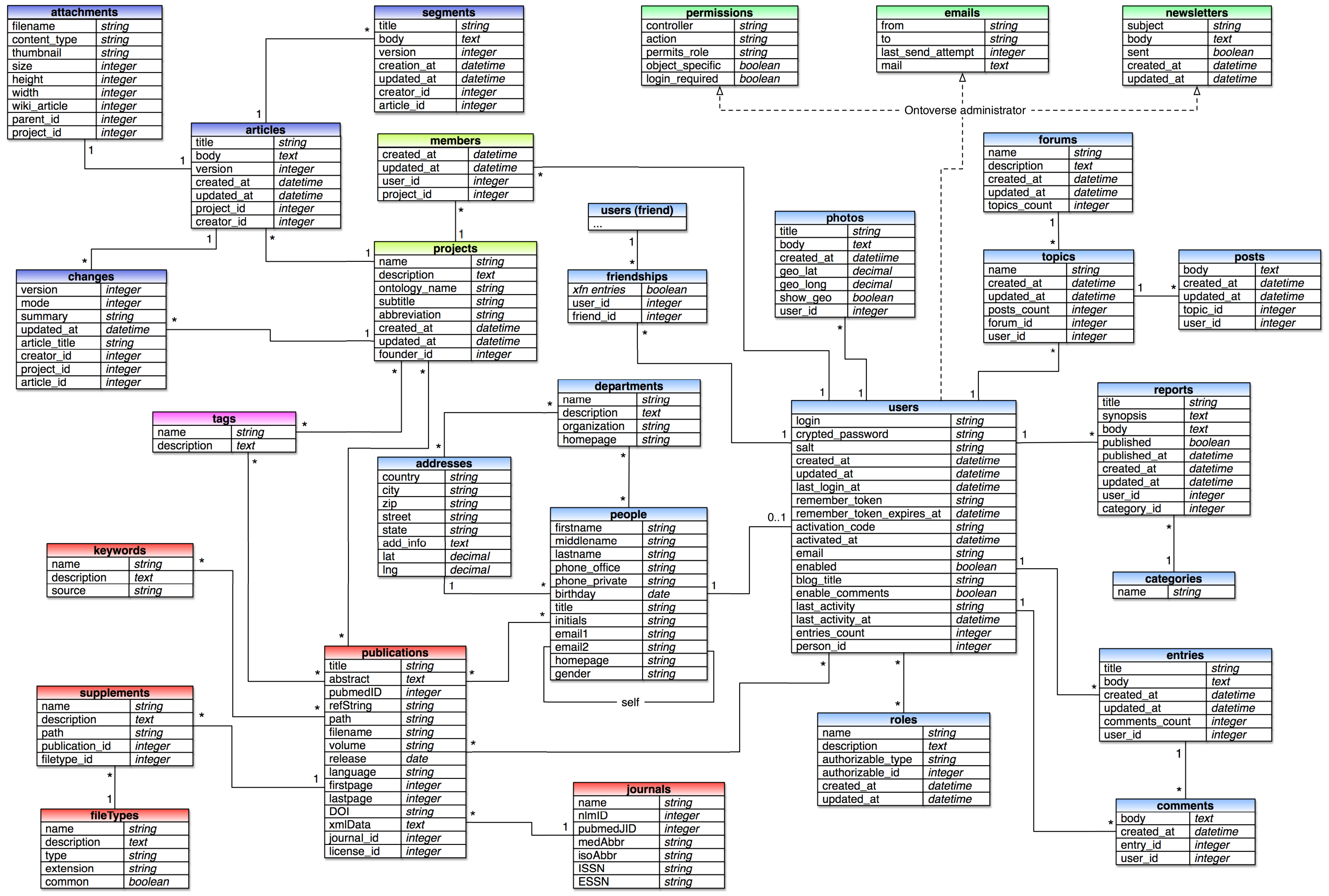


Figure A.1: Database schema of the Ontoverse ontology wiki. Tables with the same color are grouped into different parts of the architecture.

Abbreviations

ACID	Add, Change, Inquire, Delete
ANSI	American National Standards Institute
API	Application Programming Interface
BIO2Me	BioInformatics Ontology for Tools and Methods
CMS	Content Management System
CRUD	Create, Read, Update, Delete
DE	Domain expert
DL	Description Logics
DNA	Deoxyribonucleic acid
DTD	Document Type Definition
EMBL	European Molecular Biology Laboratory
GF	Gene family
GO	Gene Ontology
GUI	Graphical User Interface
HOVERGEN	HOmologous VERtebrate GENes
HSP	High-scoring Segment Pair
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IE	Information extraction
INVHOGEN	INVertebrate HOmologous GENes
ISO	International Organization for Standardization
MO	Managed Object
MOC	Managed Object Context

MOM	Managed Object Model
MSA	Multiple Sequence Alignment
MVC	Model-View-Controller
NCBI	National Center for Biotechnology Information
OA	Ontoverse administrator
OD	Ontology designer
ORSD	Ontology Requirement Specification Document
OWL	Web Ontology Language
PA	Project administrator
RDF	Resource Description Framework
RDFS	RDF Schema
REST	REpresentational State Transfer
ROA	Resource-Oriented Architecture
RSS	Really Simple Syndication
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol (originally)
SPARQL	SPARQL Protocol and RDF Query Language
SQL	Structured Query Language
SWRL	Semantic Web Rule Language
TrEMBL	Translated EMBL
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WWW	World Wide Web
WYSIWYG	What You See Is What You Get
XFN	XHTML Friendship Network
XHTML	Extensible Hypertext Markup Language
XML	Extensible Markup Language
XML-RPC	XML Remote Procedure Call
YARS	Yet Another RDF Store

Bibliography

[1] Boeckmann, B., Bairoch, A., Apweiler, R., Blatter, M.C., Estreicher, A., Gasteiger, E., Martin, M.J., Michoud, K., O'Donovan, C., Phan, I., Pilbout, S., Schneider, M. (2003) The SWISS-PROT protein knowledge base and its supplement TrEMBL in 2003. *Nucleic Acids Res.*, **31**, 365-370.

[2] Ashburner, M. *et al.* (2000) Gene Ontology: tool for the unification of biology. *Nat. Genet.*, **25**, 25-29.

[3] Camon, E., Magrane, M., Barrell, D., Binns, D., Fleischmann, W., Kersey, P., Mulder, N., Oinn, T., Maslen, J., Cox, A. *et al.* (2003) The Gene Ontology Annotation (GOA) project: implementation of GO in SWISS-PROT, TrEMBL and InterPro. *Genome Res.*, **13**, 662-672.

[4] Conesa, A., Götz, S., García-Gómez, J.M., Terol, J., Talón, M., Robles, M. (2005) Blast2GO: a universal tool for annotation, visualization and analysis in functional genomics research. *Bioinformatics*, **21**, 3674-3676.

[5] Zehetner, G. (2003) OntoBlast function: From sequence similarities directly to potential functional annotations by ontology terms. *Nucleic Acids Res.*, **31**, 3799-3803.

[6] Groth, D., Lehrach, H., Hennig, S. (2004) GOblet: a platform for Gene Ontology annotation of anonymous sequence data. *Nucleic Acids Res.*, **32**, W313-W317.

[7] Lord, P., Stevens, R., Brass, A., Goble, C. (2003) Investigating semantic similarity measures across the Gene Ontology: the relationship between sequence and annotation. *Bioinformatics*, **19(10)**, 1275-1283.

[8] Jakonienė, V., Rundqvist, D., Lambrix, P. (2006) A method for similarity-based grouping of biological data. *3rd International Workshop on Data Integration in the Life Sciences*, LNBI 4075, 136-151.

[9] Stevens, R., Goble, C.A., Bechhofer, S. (2000) Ontology-based Knowledge Representation for Bioinformatics. *Briefings in Bioinformatics*, **1(4)**, 398-416.

[10] Cheng, J., Sun, S., Tracy, A., Hubbell, E., Morris, J., Valmeekam, V., Kimbrough, A., Cline, M.S., Liu, G., Shigeta, R., Kulp, D., Siani-Rose, M.A. (2004) NetAffx

Gene Ontology Mining Tool: a visual approach for microarray data analysis. *Bioinformatics*, **20**, 1462-1463.

[11] Mao, X., Cai, T., Olyarchuk, J.G., Wei, L. (2005) Automated genome annotation and pathway identification using the KEGG Orthology (KO) as a controlled vocabulary. *Bioinformatics*, **21(19)**, 3787-3793.

[12] Frankewitsch, T., Prokosch, U. (2001) Navigation in medical Internet image databases. *Med Inform Internet Med.*, **26(1)**, 1-15.

[13] Edwards, J.L., Lane, M.A., Nielsen, E.S. (2000) Interoperability of Biodiversity Databases: Biodiversity on Every Desktop. *Science*, **289**, 2312-2314.

[14] Paulsen, I., Mainz, D., Weller, K., Mainz, I., Kohl, J., von Haeseler, A. (2007) Ontoverse: Collaborative Knowledge Management in the Life Sciences Network. *In: Proceedings of the Germany eScience Conference 2007*, Max Planck Digital Library, ID 316588.0.

[15] Paulsen, I., von Haeseler, A. (2006) Invhogen: a database of homologous invertebrate genes. *Nucleic Acids Res.*, **34**, D349-D353.

[16] Gruetter, R., Eikemeier, C. (2004) Applying the Semantic Web to clinical process. *Proceedings of 49. Jahrestagung der Deutschen Gesellschaft für Medizinische Informatik, Biometrie und Epidemiologie, Innsbruck, Austria (26th–30th September 2004).*

[17] Berners-Lee, T., Hendler, J., Lassila, O. (2001, May) The Semantic Web. *Scientific American*, pp. 28-37.

[18] Decker, S., Melnik, S. (2000, September/October) The Semantic Web: The roles of XML and RDF. *IEEE Internet Computing.*

[19] Cardoso, J. (Ed.) (2007) *Semantic Web Services: Theory, Tools and Applications.* Information Science Reference. Hershey, New York.

[20] Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., Dean, M. (2003) SWRL: A semantic web rule language combining OWL and RuleML. Available at `http://www.daml.org/2003/11/swrl/`

[21] Colomb, R. (2005) *Ontology and the semantic web study book* (Vol. 1). Brisbane: University of Queensland.

[22] McGuinness, D.L., Van Harmelen, F. (Eds.) (2004, February) *OWL Web ontology language overview W3C recommendation.* Retrieved December 20, 2005, from `http://www.w3.org/TR/owl-features/`

[23] Neches, R., Fikes, R.E., Finin, T., Gruber, T.R., Senator, T., Swartout, W.R. (1991) Enabling technology for knowledge sharing. *AI Magazine*, **12(3)**, 36-56.

[24] Gruber, T.R. (1993) A translation approach to portable ontologies. *Knowledge Acquisition*, **5(2)**, 199-220.

[25] Guarino, N., Giaretta, P. (1995) *Ontologies and knowledge bases: Towards a terminological clarification.* In *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing*, Mars N (ed). IOS Press: Amsterdam, pp. 25-32.

[26] Sheth, A. (2003, July) Semantic metadata for enterprise information integration. *DM Review.*

[27] Jasper, R., Uschold, M. (1999) *A framework for understanding and classifying ontology applications.* Paper presented at the IJCAI99 Workshop on Ontologies and Problem-Solving Methods.

[28] Fensel, D. (2001) *Ontologies: Silver bullet for knowledge management and electronic commerce.* Berlin: Springer-Verlag. Retrieved October 24, 2006, from `http://www.cs.vu.nl/ dieter/ftp/paper/silverbullet.pdf`

[29] Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (Eds.). (2003) *The description logic handbook.* Cambridge University Press.

[30] Gómez-Pérez, A. (2004) Ontology evaluation. In *Handbook on Ontologies*, Volume 10 of *International Handbooks on Information Systems*, chapter 13. Staab S, Studer R (eds). Springer: pp. 251-274.

[31] Li, L., Horrocks, I. (2004) A software framework for matchmaking based on Semantic Web technology. *International Journal of Electronic Commerce*, **8(4)**, 39-60.

[32] Lacy, L.W. (2005) *OWL: Representing Information Using the Web Ontology Language.* Trafford Publishing.

[33] Bard, J.B., Rhee, S.Y. (2004) Ontologies in biology: design, applications and future challenges. *Nat Rev Genet.*, **5(3)**, 213-222.

[34] Kumar, A., Smith, B. (2004) On controlled vocabularies in bioinformatics: A case study in gene ontology. *Drug Discovery Today: BIOSILICO*, **2**, 246-252.

[35] Bodenreider, O., Aubry, M., Burgun, A. (2005) *Non-lexical approaches to identifying associative relations in the gene ontology.* Paper presented at the Pacific Symposium on Biocomputing, Hawaii. World Scientific.

[36] Wroe, C.J., Stevens, R.D., Goble, C.A., Ashburner, M. (2003) A methodology to migrate the gene ontology to a description logic environment using DAML+OIL. *Pac. Symp. Biocomput.* pp. 624-635.

[37] MGED. (2005) *Microarray gene expression data society.* Retrieved October 24, 2006, from `http://www.mged.org/`

[38] Stoeckert, C.J., Causton, H.C., Ball, C.A. (2002) Microarray databases: Standards and ontologies. *Nature Genetics*, **32**, 469-473.

[39] Stevens, R., Baker, P., Bechhofer, S., Ng, G., Jacoby, A., Paton, N.W., Goble, C.A., Brass, A. (2001) TAMBIS: Transparent Access to Multiple Bioinformatics Information Sources. *Bioinformatics*, **16(2)**, 184-186.

[40] Hillegass, A. (2004) *Cocoa Programming for Mac OS X, Second Edition.* Addison-Wesley.

[41] Cox, B., Novobilski, A. (1991) *Object-Oriented Programming: An Evolutionary Approach, Second Edition.* Addison-Wesley.

[42] Apple Inc. (2006) *Apple's Developer Connection: Developing with Core Data.* `http://developer.apple.com/macosx/coredata.html`

[43] Thomas, D., Fowler, C., Hunt, A. (2005) *Programming Ruby: The Pragmatic Programmers' Guide, Second Edition.* The Pragmatic Programmers, LLC.

[44] Thomas, D., Heinemeier Hansson, D. (2006) *Agile Web Development with Rails, Second Edition.* The Pragmatic Programmers, LLC.

[45] Garrett, J.J. (2005, February 18) *Ajax: A New Approach to Web Applications.* Adaptive Path, LLC.

[46] Black, D.A. (2006) *Ruby for Rails.* Manning Publications Co.

[47] Fielding, R.T. (2000) Architectural styles and the design of network-based software architectures. *PhD Thesis*, University of California, Irvine.

[48] Richardson, L., Ruby, S. (2007) *RESTful Web Services.* O'Reilly Media, Inc.

[49] Bernal, A., Ear, U., Kyrpides, N. (2001) Genomes OnLine Database (GOLD): a monitor of genome projects world-wide. *Nucleic Acids Res.*, **29**, 126-127.

[50] Eisenberg, D., Marcotte, E.M., Xenarios, I., Yeates, T.O. (2000) Protein function in the post-genomic era. *Nature*, **405**, 823-826.

[51] Henikoff, S., Greene, E.A., Pietrokovski, S., Bork, P., Attwood, T.K., Hood, L. (1997) Gene families: the taxonomy of protein paralogs and chimeras. *Science*, **278**, 609-614.

[52] Orengo, C.A., Todd, A.E., Thornton, J.M. (1999) From protein structure to function. *Curr. Opin. Struct. Biol.*, **9**, 374-382.

[53] Heger, A., Holm, L. (2000) Towards a covering set of protein family profiles. *Prog. Biophys. Mol. Biol.*, **73**, 321-337.

[54] Fitch, W.M., Margoliash, E. (1970) The usefulness of amino acid and nucleotide sequences in evolutionary studies. *Evolutionary Biology*, **2**, 67-109.

[55] Duret, L., Perrière, G., Gouy, M. (1999) HOVERGEN: database and software for comparative analysis of homologous vertebrate genes. In Bioinformatics and Systems, Letovsky, S. (ed.), Kluwer Academic Publishers, Boston, pp. 13-29.

[56] Perrière, G., Duret, L., Gouy, M. (2000) HOBACGEN: database system for comparative genomics in bacteria. *Genome Res.* **10**, 379-385.

[57] Bairoch, A., Apweiler, R., Wu, C.H., Barker, W.C., Boeckmann, B., Ferro, S., Gasteiger, E., Huang, H., Lopez, R., Magrane, M., Martin, M.J., Natale, D.A., O'Donovan, C., Redaschi, N., Yeh, L.S. (2005) The Universal Protein Resource (UniProt). *Nucleic Acids Res.*, **33**, D154-D159.

[58] Kanz, C., Aldebert, P., Althorpe, N., Baker, W., Baldwin, A., Bates, K., Browne, P., van den Broek, A., Castro, M., Cochrane, G., Duggan, K., Eberhardt, R., Faruque, N., Gamble, J., Diez, F.G., Harte, N., Kulikova, T., Lin, Q., Lombard, V., Lopez, R., Mancuso, R., McHale, M., Nardone, F., Silventoinen, V., Sobhany, S., Stoehr, P., Tuli, M.A., Tzouvara, K., Vaughan, R., Wu, D., Zhu, W., Apweiler, R. (2005) The EMBL Nucleotide Sequence Database. *Nucleic Acids Res.*, **33**, D29-D33.

[59] Altschul, S.F., Madden, T.L., Schäffer, A.A., Zhang, J., Zhang, Z., Miller, W., Lipman, D.J. (1997) Gapped BLAST and PSI-BLAST: A new generation of protein database search programs. *Nucleic Acids Res.*, **25**, 3389-3402.

[60] Wootton, J.C., Federhen, S. (1996) Analysis of compositionally biased regions in sequence databases. *Methods Enzymol.*, **266**, 554-571.

[61] Henikoff, S., Henikoff, J.G. (1992) Amino acid substitution matrices from protein blocks. *Proc Natl Acad Sci.*, **89**, 10915-10919.

[62] Nei, M. (1996) Phylogenetic analysis in molecular evolutionary genetics. *Annu. Rev. Genet.*, **30**, 371-403.

[63] Chenna, R., Sugawara, H., Koike, T., Lopez, R., Gibson, T.J., Higgins, D.G., Thompson, J.D. (2003) Multiple sequence alignment with the Clustal series of programs. *Nucleic Acids Res.*, **31**, 3497-3500.

[64] Vinh, L.S., von Haeseler, A. (2004) IQPNNI: Moving fast through tree space and stopping in time. *Mol. Biol. Evol.*, **21**, 1565-1571.

[65] Hedges, S.B. (2002) The origin and evolution of model organisms. *Nature Genet.*, **3**, 838-849.

[66] Fraser, A.G., Marcotte, E.M. (2004) A probabilistic view of gene function. *Nat. Genet.*, **36**, 559-564.

[67] Wheeler, D.L., Chappey, C., Lash, A.E., Leipe, D.D., Madden, T.L., Schuler, G.D., Tatusova, T.A., Rapp, B.A. (2000) Database resources of the National Center for Biotechnology Information. *Nucleic Acids Res.*, **28**, 10-14.

[68] Benson, D.A., Karsch-Mizrachi, I., Lipman, D.J., Ostell, J., Rapp, B.A., Wheeler, D.L. (2000) GenBank. *Nucleic Acids Res.*, **28**, 15-18.

[69] May, R.M. (2000) The Dimensions of Life on Earth. In Raven, P.H. (ed.) *Nature and Human Society: The Quest for a Sustainable World*, Chapter 1 Defining Biodiversity. The National Academy of Sciences, Washington, pp. 30-45.

[70] Holm, L., Sander, S. (1996) Mapping the protein universe. *Science*, **273**, 595-602.

[71] Berman, H.M., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T.N., Weissig, H., Shindyalov, I.N., Bourne, P.E. (2000) The Protein Data Bank. *Nucleic Acids Res.*, **28**, 235-242.

[72] Geer, L.Y., Domrachev, M., Lipman, D.J., Bryant, S.H. (2002) CDART: protein homology by domain architecture. *Genome Res.*, **12**, 1619-1623.

[73] Servant, F., Bru, C., Carrere, S., Courcelle, E., Gouzy, J., Peyruc, D., Kahn, D. (2002) Prodom: automated clustering of homologous domains. *Brief Bioinform.*, **3**, 246-251.

[74] Bateman, A., Birney, E., Cerruti, L., Durbin, R., Etwiller, L., Eddy, S.R., Griffiths-Jones, S., Howe, K.L., Marshall, M., Sonnhammer, E.L. (2002) The Pfam protein families database. *Nucleic Acids Res.*, **30**, 276-280.

[75] Mulder, N.J., Apweiler, R., Attwood, T.K., Bairoch, A., Barrell, D., Bateman, A., Binns, D., Biswas, M., Bradley, P., Bork, P., *et al.* (2003) InterPro, progress and status in 2005. *Nucleic Acids Res.*, **33**, D201-D205.

[76] Burges, C.J.C. (1998) A tutorial on Support Vector Machine for pattern recognition. *Data Min. Knowl. Disc.*, **2**, 121-167.

[77] Wolstencroft, K., McEntire, R., Stevens, R., Tabernero, L., Brass, A. (2005) Constructing ontology-driven protein family databases. *Bioinformatics*, **21**, 1685-1692.

[78] Godfray, H.C.J. (2002) Challenges for taxonomy. *Nature*, **417**, 17-19.

[79] Hennig, S., Groth, D., Lehrach, H. (2003) Automated Gene Ontology annotation for anonymous sequence data. *Nucleic Acids Res.*, **31**, 3712-3715.

[80] Khan, S., Situ, G., Decker, K., Schmidt, C.J. (2003) GoFigure: automated Gene Ontology annotation. *Bioinformatics*, **19**, 2484-2485.

[81] Martin, D.M., Berriman, M., Barton, G.J. (2004) GOtcha: a new method for prediction of protein function by the annotation of seven genomes. *BMC Bioinformatics*, **5**, 178.

[82] Young, A., Whitehouse, N., Cho, J., Shaw, C. (2005) OntologyTraverser: an R package for GO analysis. *Bioinformatics*, **21**, 275-276.

[83] Lee, J.S., Katari, G., Sachidanandam, R. (2005) GObar: a gene ontology based analysis and visualization tool for gene sets. *BMC Bioinformatics*, **6**, 189.

[84] Ye, J., Fang, L., Zheng, H., Zhang, Y., Chen, J., Zhang, Z., Wang, J., Li, S., Li, R., Bolund, L., Wang, J. (2006) WEGO: a web tool for plotting GO annotations. *Nucleic Acids Res.*, **34**, W293-W297.

[85] Studer, R., Benjamins, V.R., Fensel, D. (1998) Knowledge Engineering. Principles and Methods. *In: IEEE Transactions on Data and Knowledge Engineering*, **25(1-2)**, 161-197.

[86] Rubin, D.L., Lewis, S.E., Mungall, C.J., Misra, S., Westerfield, M., Ashburner, M., Sim, I., Chute, C.G., Solbrig, H., Storey, M.A., Smith, B., Day-Richter, J., Noy, N.F., Musen, M.A. (2006) National Center for Biomedical Ontology: advancing biomedicine through structured organization of scientific knowledge. *OMICS*, **10(2)**, 185-198.

[87] Mainz, I. (2006) Entwicklung einer Prototypontologie für bioinformatische Werkzeuge. *Bachelorarbeit*, Heinrich-Heine-Universität Düsseldorf.

[88] Fernández-López, M. (2001) Overview of methodologies for building ontologies. In *Proceedings of the IJCAI-99 Workshop on Ontologies*.

[89] Sure, Y. (2002) A Tool-supported Methodology for Ontology-based Knowledge Management, submitted to ISMIS 2002, *Methodologies for Intelligent Systems*.

[90] Noy, N.F., Chugh, A., Liu, W., Musen, M.A. (2006) A Framework for Ontology Evolution in Collaborative Environments. *5th International Semantic Web Conference*, Athens, GA.

[91] Das, A., Wu, W., McGuinness, D. (2001, August) Industrial strength ontology management. In *Proceedings of the First Semantic Semantic Web Working Symposium*, SWWS-01, Stanford, USA.

[92] Domingue, J. (1998) Tadzebao and WebOnto: Discussing, Browsing, and Editing Ontologies on the Web. *11th Knowledge Acquisition for Knowledge-Based Systems Workshop*, April 18th-23rd. Banff, Canada.

[93] McGuinness, D.L., Fikes, R., Rice, J., Wilder, S. (2000, July) The Chimaera Ontology Environment. *Proceedings of the The Seventeenth National Conference on Artificial Intelligence*, Austin, Texas.

[94] Farquhar, A., Fikes, R., Rice, J. (1996) The Ontolingua Server: a Tool for Collaborative Ontology Construction. *Technical report*, Stanford KSL, 96-126.

[95] Bao, J., Honavar, V. (2004) Collaborative Ontology Building with Wiki@nt. A Multi-agent Based Ontology Building Environment. In *Proceedings of the 3rd International Workshop on Evaluation of Ontology-based Tools (EON)*, Hiroshima 2004, 1-10.

[96] Hepp, M., Bachlehner, D., Siorpaes, K. (2005) OntoWiki — Community-driven Ontology Engineering and Ontology Usage based on Wikis. *Proceedings of the 2005 International Symposium on Wikis (WikySym)*, San Diego.

[97] Pinto, H.S., Staab, S., Tempich, C. (2004) DILIGENT. Towards a fine-grained methodology for Distributed, Loosely-controlled and Evolving Engineering of Ontologies. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI)*, 393-397.

[98] Zacharias, V., Braun, S. (2007) SOBOLEO — Social Bookmarking and Lightweight Ontology Engineering. In *Workshop on Social and Collaborative Construction of Structured Knowledge (CKC), 16th International World Wide Web Conference (WWW 2007)*, Banff, Alberata, Canada

[99] Fensel, D. (2004) *Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce*. Springer-Verlag.

[100] Baclawski, K., Niu, T. (2006) *Ontologies for Bioinformatics*. The MIT Press.

[101] Wilkinson, K., Sayers, C., Kuno, H.A., Reynolds, D. (2003, September) Efficient RDF Storage and Retrieval in Jena2. In *Proceedings of SWDB03, 1st International Workshop on Semantic Web and Databases, Co-located with VLDB 2003, Berlin*, 131-150.

[102] Beckett, D. (2002) The design and implementation of the Redland RDF application framework. *Computer Networks*, **39(5)**, 577-588.

[103] Harth, A., Decker, S. (2005) Optimized index structures for querying RDF from the web. In *LA-WEB*.

[104] Gelernter, D. (1985) Generative communication in Linda. *ACM Trans. Program. Lang. Syst.*, **7(1)**, 80-112.

[105] Shneiderman B. (1996) The eye have it: A task by data type taxonomy for information visualizations. In *Proc. Visual Languages*.

[106] Leung, Y., Apperley, M. (1994) A review and taxonomy of distortion-oriented presentation techniques. In *Proc. Human Factors in Computing Systems CHI '94 Conf.*, Boston, MA, 126-160.

[107] Jenkins, B. (1997, September) Hash Functions. *Dr. Dobb's Journal*, 1-5.

[108] Sundheim, B.M. (1992) Overview of the fourth Message Understanding evaluation and Conference. In *Proceedings of the Fourth Message Understanding Conference (MUC-4)* (pp. 3-21). San Mateo, CA: Morgan Kaufmann.

[109] Grishman, R., Sundheim, B. (1996) Message Understanding Conference 6: A brief history. In *Proceedings of the* 16^{th} *International Conference on Computational Linguistics* (pp. 466-471). San Mateo, CA: Morgan Kaufmann.

[110] Cunningham, H. (1997) *Information Extraction: A User Guide*. Research memo CS-97-02. Sheffield: University of Sheffield, ILASH.

[111] Malzahn, N., Weinbrenner, S., Hüsken, P., Ziegler, J., Hoppe, H.U. (2007) Collaborative Ontology Development — Distributed Architecture and Visualization. *In: Proceedings of the Germany eScience Conference 2007*, Max Planck Digital Library, ID 315470.0.

Die hier vorgelegte Dissertation habe ich eigenständig und ohne unerlaubte Hilfe angefertigt. Die Dissertation wurde in der vorgelegten oder in ähnlicher Form noch bei keiner anderen Institution eingereicht. Ich habe bisher keine erfolglosen Promotionsversuche unternommen.

Düsseldorf, den 31. Oktober 2007 Ingo Paulsen