View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Ghent University Academic Bibliography

ir. Ward Blondé

Metarel, an ontology facilitating advanced querying of biomedical knowledge

Thesis submitted in fulfillment of the requirements for the degree of

Doctor of Applied Biological Sciences

Academic year 2011-2012

Supervisors:	Prof. Dr. Bernard De Baets Department of Mathematical Modelling, Statistics and Bioinformatics Ghent University, Belgium Prof. Dr. Martin Kuiper Department of Biology Norwegian University of Science and Technology, Trondheim, Norway
Examination committee:	Prof. Dr. Ir. Koen Dewettinck (Chairman) Prof. Dr. Godelieve Gheysen (Secretary) Prof. Dr. Med. Stefan Schulz Prof. Dr. Guy De Tré Prof. Dr. Ir. Tim De Meyer Prof. Dr. Martine De Cock Prof. Dr. Peter Dawyndt Dr. Erick Antezana
Dean:	Prof. Dr. Ir. Guido Van Huylenbroeck
Rector:	Prof. Dr. Paul Van Cauwenberge

ir. Ward Blondé

Metarel, an ontology facilitating advanced querying of biomedical knowledge

Thesis submitted in fulfillment of the requirements for the degree of

Doctor of Applied Biological Sciences

Academic year 2011-2012

Dutch translation of the title:

Metarel, een ontologie voor doorgedreven bevraging
van biomedische kennis

Please refer to this work as follows:
Ward Blondé (2012). *Metarel, an ontology facilitating advanced querying of biomedical knowledge*, PhD Thesis, Department of Mathematical Modelling, Statistics and Bioinformatics, Ghent University, Ghent, Belgium.

Cover:
Linking Open Data cloud diagram, by Richard Cyganiak and Anja Jentzsch.
http://lod-cloud.net/

ISBN 978-90-5989-532-4

The author and the supervisor give the authorization to consult and to copy parts of this work for personal use only. Every other use is subject to the copyright laws. Permission to reproduce any material contained in this work should be obtained from the author.

Dankwoord

Het schrijven van deze thesis en de publicaties die eraan voorafgaan is een werk van lange adem geweest, waarbij de steun van vele anderen onontbeerlijk is gebleken. In een tijdsspanne van vijf jaar heb ik daarbij aan vier verschillende universitaire instituten kunnen verblijven. Dat ik al die tijd aan mijn doctoraat heb kunnen blijven werken, heb ik te danken aan de ruimdenkendheid van mijn promotoren, Bernard De Baets en Martin Kuiper, van mijn mentor in Graz, Stefan Schulz en niet in het minst ook van mijn vader, Johan Blondé, die intussen is overleden, maar die mij op weg geholpen heeft als werknemer van Blondé Engineering. Ik ben hen uiteraard ook dank verschuldigd voor alle hulp en goede raad die ze mij voortdurend hebben gegeven. De leden van de jury, die deze thesis grondig hebben doorgelezen en beoordeeld, wil ik bedanken voor de vele nuttige aanmerkingen die tot de verbetering van deze thesis hebben geleid.

De vele collega's die ik heb gehad tijdens deze omzwervingen wil ik allemaal bedanken. Daarbij denk ik in het bijzonder aan Erick Antezana en Vladimir Mironov, die ik in de VIB in Zwijnaarde heb leren kennen en waarmee ik sindsdien altijd goed heb kunnen samenwerken. Ook Mikel Egaña was erbij toen het onderwerp van mijn doctoraat eind 2006 werd uitgetekend. Kenny Billiau wil ik bedanken voor zijn bereidheid om mij wegwijs te maken in Unix en Linux. De vele buitenlandse doctoraatsstudenten aan de VIB mag ik hier niet vergeten, en de 'Ghent friends' in het bijzonder, want zij hebben het verlangen bij mij ontwaakt om zelf ook de wereld in te trekken en niet in het kleine België te blijven hangen. Martin en Steven Vercruysse hebben mij heel concreet het goede voorbeeld gegeven door naar Trondheim te verhuizen. Later werd ik er heel gastvrij ontvangen door Martin en zijn gezin. Tijdens mijn verblijf daar heb ik Aravind Venkatesan leren kennen, wiens huwelijk in Indië later een sprookjesachtige ervaring werd voor mij. Ook de vele onderzoekers die ik leerde kennen op wetenschappelijke conferenties waren een inspiratiebron voor mij. Sine Zambach

dank ik voor haar gastvrijheid en samenwerking in Kopenhagen.

De collega's aan de Coupure in Gent wil ik bedanken voor de vele gezellige activiteiten en de hechte groepsgeest, waar ik een tijdje deel van mocht uitmaken. Ze zijn met teveel om ze allemaal op te noemen. Karel, Stijn en Wim waren er altijd om een praatje te maken. Met Tarad, Engin, Abhishek en Yingjie kon ik diepzinnige gesprekken voeren over de Islam en de snelgroeiende Aziatische culturen. Zij hebben mij nog meer zin gegeven om de wereld te ontdekken. Dankzij al deze mensen moest ik niet lang aarzelen om aan de slag te gaan bij Stefan in Oostenrijk. Gelukkig waren Cati Martinez-Costa en André Andrade solidair met deze beslissing en kan ik met mijn bureaugenoten in het Engels van gedachten wisselen. Het verblijf aan al deze wetenschappelijke instituten werd natuurlijk ook mogelijk gemaakt door vele secretariaatsmensen, IT-specialisten en onderhoudspersoneel.

Mijn vrienden wil ik bedanken omwille van de inspirerende voorbeelden die ze zijn geweest, zowel degene die ik ken sinds de periode van 't Sint Lodewijks, de leden van de 'KKK', als die van de ingenieursjaren in Gent. Sam, wiens thesis hier nu naast mij ligt, maakte mij zelfs peter van zijn zoontje Nolan.

Tenslotte wil ik mijn familie bedanken voor alle morele steun die ze mij altijd hebben gegeven: papa, mama, Griet, Klaas en ook de gezinnetjes die jullie hebben gesticht. Hoe jammer papa dat je er niet meer bent op dit moment, maar jouw geloof in mij blijft ook na je dood nog verderbestaan.

Ward
Graz, 11 juni 2012

Contents

List of Figures

List of Tables

List of Symbols and Acronyms

$\wedge$	and
$\forall$	for all
$\top$	everything
$\Leftrightarrow$	if and only if
$\sqcap$	intersection
$\in$	is element of
$=$	is equal to
$\equiv$	is equivalent to
$\sqsubseteq$	is subset of or equal to
$\neg$	not (in Logic), or complement (in Set Theory)
!	not (in SPARQL)
$\bot$	nothing
$\vee$	or
$\exists$	there exists
$\sqcup$	union
?x (or $x)	variable x (in SPARQL)
3D	3-dimensional
A	set of relation arcs
A	class A
acc	accession number
ALC	Attributive Concept Language with Complements
alt	alternative
API	Application Programming Interface
ART	Average Response Time
ASCII	American Standard Code for Information Interchange
At	*Arabidopsis thaliana*
avg	average
B	intermediate or target class in a chain of relations
BFO	Basic Formal Ontology
bgw	BioGateway

Bio1	Biomedical query 1
BioMetarel	Metarel-annotated ontology of biomedical relation types
Biorel	ontology of biomedical relation types
BLAST	Basic Local Alignment Search Tool
BMC	BioMed Central
BSBM	Berlin SPARQL Benchmark
BSPO	Biomedical Spatial Ontology
$\mathtt{c}$	continuant instance
$\mathtt{C}$	continuant class
C	set of class nodes
C	third class in a chain (reasoning)
CBD	Computational Biology Division
CCO	Cell Cycle Ontology
c_d	class that represents a domain
CPU	Central Processing Unit
c_r	class that represents a range
CSV	Comma-Separated Values
CV	Curriculum Vitae
D	set of nodes
D5.4	European project deliverable
DAG	Directed Acyclic Graph
DAML	DARPA Agent Markup Language
db (or DB)	Database
dbname	Database name
def	definition
DL	Description Logic
DM	Data Management
DNA	desoxyribonucleic acid
DOT	DOT language for graph representation
DS	Direct Semantics
$\mathcal{E}$	set of couples E
EL	basic Description Logic $\mathcal{EL}$
EU	European Union
Exp.	Explicit knowledge
EXPSPACE	Exponential memory space complexity
EXPTIME	Exponential time complexity
$Ext(c)$	set of instance nodes of c
FOL	First Order Logic
FP6	Sixth Framework Programme
g	graph
Geom.	Geometrical
GB	Gigabyte
GCI	General Concept Inclusion

GHz	Gigahertz
GML	Graph Modelling Language
GO	Gene Ontology
GOA	Gene Ontology Annotations
Hs	*Homo sapiens*
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
I	set of instance nodes
id (or ID)	identifier
Imp.	Implicit knowledge
irt	invertible relation type
IRI	Internationalized Resource Identifier
is_a	'is a' – conjugation of verb 'to be' plus indefinite pronoun
ISO/IEC	International Organization for Standardization/International Electrotechnical Commission
ISS	International Space Station
KB	Knowledge Base
kg	kilogram
KERMIT	Knowledge Extraction, Representation and Management by means of Intelligent Techniques
KM	Knowledge Management
KR	Knowledge Representation
L	Linear time complexity
LOD	Linked Open Data
LUBM	Lehigh University Benchmark
Max	Maximum
MB	Megabyte
MDG	MultiDiGraph
Metaonto	ontology for metadata about ontologies
Metarel	ontology for relations and metarelations
mrl	Metarel
n1	name 1
N3	Notation 3 – a non-XML-based syntax for RDF
NCBI	National Center for Biotechnology Information
NCBO	National Center for Biomedical Ontology
NP	Non-deterministic Polynomial time complexity
NTNU	Norwegian University of Science and Technology
o	object
O	ontology modelled as MDG
OBO	Open Biomedical Ontologies
OBOF	Open Biomedical Ontology Format
OBO_REL	OBO relationship type
OIL	Ontology Inference Layer

OLS	Ontology Lookup Service
ONTO1	Ontological query 1
ONTO-PERL	Perl software package for ontology management
OWL	Web Ontology Language
p	process instance
P	process class
p	predicate
P (ontology)	process
P (complexity)	Polynomial time complexity
PPI	protein-protein interaction
PSPACE	Polynomial memory space
PubMed	Biomedical publication database
Q1	Query 1
r	multi-arc (relation)
R	set of multi-arcs, or relation type R
$\mathcal{R}$	mathematical, binary relation
R (or R)	relation type R
(R)	Registered Trademark
R1	Rule 1
RBS	RDF-Based Semantics
RDBMS	Relational Database Management System
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
regex	regular expression
rel	relation type
RIA	Role Inclusion Axiom
RIF	Rule Interchange Format
RO	Relationship Ontology
RPM	Revolutions Per Minute
rt	relation type
RTC	Relation Type Class
s	subject
S	relation type S
SAS	Serial Attached SCSI (hard drive)
Sc	*Saccharomyces cerevisiae*
SDB	SPARQL Database
SGML	Standard Generalized Markup Language
SNOMED CT	Systematized Nomenclature of Medicine - Clinical Terminology
Sp (or SCHPO)	*Schizosaccharomyces pombe*
SPARQL	SPARQL Protocol And RDF Query Language
SPARUL	SPARQL/Update
SQL	Structured Query Language

SSB	Semantic Systems Biology
$Sub(c)$	set of subclass nodes of c
sup	superclasses
SWAT4LS	Semantic Web Applications and Tools for the Life Sciences
SWISS-PROT	protein database developed by EBI and Swiss Institute SIB
SWRL	Semantic Web Rule Language
syn	synonym
$\mathtt{t}$	time t
t	relation type
T	set of relation types, or relation type T
T	taxon (term in a taxonomy about organisms)
tc	total closures
TDB	Triple Database
Turtle	Terse RDF triple language
U	upper level term
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URN	Uniform Resource Name
UTF8	8-bit Unicode Transformation Format
$\mathcal{V}$	set of elements V
W3C	World Wide Web Consortium
www	World Wide Web
x	element x
XML	Extensible Markup Language
XPath	XML Path Language
XQuery	XML Query Language
xref	cross-reference
y	element y
ZF	Zermelo-Fraenkel set theory

Chapter 1

Guide for the readers

This thesis describes some advances made in a scientific domain that belongs both to Computer Science and to the Life Sciences. More precisely, it could be called 'Knowledge Management in the Life Sciences'. The advances described are of different natures. Whereas the theoretical and technological advances described are directly related to the pillar of Computer Science, those that relate to the Life Sciences are the advances of a supportive science, that can assist life scientists in their work.

This short introductory chapter will guide the reader through the other chapters, seven in total, that constitute the contents of the thesis. The next two chapters, 2 and 3, are also introductory. The four following chapters, 4, 5, 6 and 7, describe the above-mentioned scientific advances in detail, and refer each to one or more scientific publications. Chapter 8 contains the conclusions of the thesis, whereas Chapter 9 provides a summary of the thesis in Dutch.

The bibliography, following the Dutch summary, is in turn followed by some appendices with code that is useful for computer scientists and ontologists who want to look into code about Metarel and reasoning. The last appendix presents the CV of the author and includes the publications that are relevant to chapters 4 to 7.

Chapter 2 introduces the field of Knowledge Management in the Life Sciences and presents its state of the art. It acknowledges the many efforts that were done by other scientists working in this field.

The emergence of the most recent tools for Knowledge Management is described in Chapter 3. In order to introduce them properly, a short history of Logic and Ontology is also included there.

The first scientific contributions of the author, described in Chapter 4, relate to CCO, the Cell Cycle Ontology, and RDF, the Resource Descrip-

tion Framework. CCO was developed by Erick Antezana during his PhD research and co-developed at the Computational Biology Division (CBD) at Plant Systems Biology of the VIB in Ghent. The author joined this group in September 2006, at a point when CCO, including RDF exports for CCO, had already been created through ONTO-PERL, an ontology management software package developed by Antezana [1]. The contributions described stem from an involvement in the development of the DIAMONDS platform for presenting CCO. The presentation of the DIAMONDS platform and therefore also the RDF query layer in CCO's publication in Genome Biology [2] was requested in a review process and has led to the co-authorship of the author of this thesis. The visualization layer of the DIAMONDS platform is thanks to Steven Vercruysse, who donated the Java code that he had developed for an interface called Minemap. The utility of CCO was published later in Applied Ontology [3]. Chapter 4 also contains a description of the NTNU benchmark, a project led by Vladimir Mironov, which compares different RDF software solutions on the basis of CCO's library of queries that were developed and optimized independently for answering biomedical questions [4].

Metarel, an ontology for relations described in Chapter 5, stems from the author's earliest view on Knowledge Management in the Life Sciences, namely the investigation of biomedical relations in ontologies. An initial architecture developed in the OBOF format in 2007, has matured during the years and gained compatibility with the Semantic Web standards RDF and OWL. It was presented at the ICBO conference in Buffalo in 2009 [5] and a final journal publication is currently in preparation.

Chapter 6 presents BioGateway, an RDF Knowledge Base that has enabled advanced biomedical queries over integrated resources. Metarel is used in BioGateway for the integration and management of relation types that stem from different resources. Such knowledge bases may help biomedical experts in creating new hypotheses. BioGateway was presented in SWAT4LS in Edinburgh in 2008 [6] and published in BMC Bioinformatics in 2009 [7].

Finally, in Chapter 7, scientific results are presented that combine all the advances that were made in the previous chapters. A semi-automated reasoning approach through Metarel in BioGateway has created hundreds of millions of logical inferences that can be used in answers on the biomedical queries that were previously developed. These results were published in Bioinformatics in 2011 [8].

Chapter 2

Knowledge Management

2.1 Introduction

Little more than half a century after the discovery of DNA by James Watson and Francis Crick, biotechnology has matured into real life applications that sustain our health and welfare. Some major breakthroughs that were made are becoming well known to the broad public: new cures for cancer, genetically modified crops, biofuels and crime-related DNA analysis, just to name a few. This fast technological evolution goes hand in hand with an ever expanding body of detailed knowledge about the molecular machinery that the 4 billion year long evolution of life has engineered for us. When I started my PhD studies in the fall of 2006, scientists still thought that a human DNA string would contain more than 100 000 different genes. Only a few years later, the estimate was revised to around 21 000 genes. At the same time the number of proteins was revised from 100 000 to 1 million. These facts reflect a tremendous increase of knowledge about the human genome, facilitated by high-throughput methods for DNA analysis. They imply a huge challenge for the Knowledge Management engineer whose task it is to maintain clear representations of this knowledge.

Knowledge Management has become a major challenge. We cannot rely on the limited memory of scientists for knowledge storage. Large libraries with thick books are not the most practical solution either. Only computer systems can deal adequately with the quickly expanding mass of knowledge. Biomedical scientists want to have detailed and specific knowledge accessible over the Internet for lookup. They want to see it interconnected in attractive and fast lookup sites. Bioinformaticians need this

knowledge to analyze it in automated ways, and to produce newly inferred knowledge from it. The Knowledge Management engineer has some tools at hand to deal with this challenge. They are called ontologies, databases, Web standards, computable logics, controlled vocabularies, representation languages and query languages. Knowledge Management in the Life Sciences, on this practical level, is what this thesis will be about.

Before we start to take a look at the contributions of this PhD thesis to the scientific community, we have to make a tour through the landscape of Knowledge Management that was shaped before. We will do this by visiting a series of concepts that are of importance in this scientific field.

2.2 Knowledge and data

Let us start first with defining 'knowledge'. The scientific field of Epistemology has investigated the very general question 'What is knowledge'? This question is very old, starting with the dialogue between Socrates and Theaetetus mentioned in the works of Plato [9], and it has resulted in many controversial philosophical debates throughout the ages. However, within computer science, a more practical definition is used that relates knowledge with data: knowledge is data plus an interpretation of its meaning [10].

Knowledge Management (KM) is the field of study that deals with the representation, preservation, integration, translation, sharing and distribution of knowledge. It thrives by the use of world-wide standards, computer science and the Internet and it is indispensable in domains where very large amounts of heterogeneous knowledge have to be managed.

An important building block for KM is the so-called knowledge statement. Here is an example: 'insulin is produced in pancreas'. An integrated Knowledge Base (KB) will contain more knowledge about both insulin and pancreas. Such as 'insulin is a protein', 'pancreas is part of abdomen', etc.

A more precise distinction between knowledge and data can be made by separating data from meta-data. Just like in the case of data, knowledge statements can be accompanied by extra data about who created this knowledge statement. When and in which institute was it investigated? In which journal or KB was is it published first? Which synonyms exist for terms like 'insulin' and 'pancreas'? Which methods were used to generate this knowledge statement? This extra data is the meta-data. Data and meta-data can be considered as information. With the addition of an interpretation of its meaning, information becomes knowledge.

2.3 The vast amount of biomedical knowledge

The amount of knowledge in the Life Sciences is huge. Trillions of knowledge statements are needed to capture all the knowledge that is fundamental to the architecture of biological life. Let us take a view in the multi-dimensional space of biomedical knowledge.

DNA contains many genes. Many genes appear in different varieties called alleles. Closely related species have genes that are closely related (orthologous), but are still different. Most genes encode for one or multiple RNA transcripts, which can have one or multiple proteins as translation product. These proteins go through a life cycle during which they will appear in different varieties by phosphorylations, folding operations and complex formation with other gene products. It needs no further arguments that we have a huge variety of genes and proteins to name and describe in KBs. All these facts have a multiplicative effect on the number of proteins that exist. The number of related knowledge statements multiplies again considering that every protein has assigned dozens of so-called 'protein features' like mass, length, hydrophobicity, electric conductivity, etc., which may assist in analyzing them in laboratory experiments.

Each protein has a certain function in the machinery of a cell. Proteins deriving from very similar genes often have the same function, but not always. They may have interactions with specific other proteins and with specific small molecules and hormones. They operate in specific locations of the anatomy of the cell and they are involved in specific biological pathways. The presence of every protein in the cell is regulated by specific cellular processes.

Small molecules, hormones, drugs, pesticides, synthetic and natural biomolecules all fit in a theoretical space of all the chemicals that could exist in reality. Different naming systems are used in chemistry, as compared to enzymology. One of the challenges for this space of chemicals is creating an overarching system for identifying and querying them, as well as mapping synonyms (different names for the same concepts). Also this is a sub-task of KM.

Cellular processes stand in connection with environmental conditions. This is especially important for plants and crops, since they cannot regulate their conditions by hiding or migrating. Plants will react differently to the presence of sunlight, temperature, humidity, salinity, pollution, insects, etc. In addition to a certain genotype, the interaction with the environment will create a broad spectrum of possible phenotypes. Bridging

the microscopic knowledge with the macroscopic knowledge is a task of Systems Biology.

The description of the species that live on Earth is one of the older disciplines in KM, since it started long before the discovery of cells and DNA. Carolus Linnaeus published Systema Naturae [11] in 1735 in which he proposes a taxonomic classification of biological life in kingdoms, classes, orders, families and species. Millions of species have been identified today. The custom to give latin names to newly discovered species still stems from this time. English has taken over this role for the more recent knowledge domains.

The number of species increases even more by considering all the micro-organisms on our planet. Scientists keep track of massive numbers of bacterial strains that have evolved in special environmental conditions. New organisms are also created in a fast rate by genetic modification. Viruses, bacteria and other small organisms evolve spontaneously under pressure of antibiotics and pesticides.

Another dimension of biological knowledge constitutes the anatomy of species. Many anatomical features are shared across many species, others are species-specific. Certain species may even lack anatomical features, such as human lacking a tail, even though they are primates.

Diseases are relevant to nearly all of the mentioned knowledge domains. They are often specific to species and anatomical parts. Specific proteins and cellular processes play a role in their development and the pharmaceutical industry investigates which molecules may cure or even cause the disease.

Knowledge Management is indispensable to capture the vast amount of knowledge that is continuously generated in all these subdomains of the Life Sciences.

2.4 Knowledge resources in the Life Sciences

Numerous scientists have contributed to Knowledge Management in the Life Sciences for many decades. Libraries with paper books have turned into relational databases some thirty years ago, resulting in countlessly many databases that are still popular today. Many of them cover a very specialized domain, like FLYBASE [12], a database with the genome of the fruit fly, p53 [13], a database on the protein type p53, and PolygenicPathways [14], a database about the genes and the risk factors that are involved

in Alzheimer's disease, bipolar disorder and schizophrenia. Within genome biology, databases are called broad if they apply to many species, like p53, and deep if they apply to a single species, like FLYBASE. Some databases hold information about storage systems, called biobanks [15], that store scientifically interesting biological samples, like StrainInfo [16], which integrates knowledge about microbial strains, or the Medical Biobank of the Umeå University Hospital in Sweden [17].

Even for similar content matter, different databases may vary in size, quality and provenance (the origin of the data). Especially the size can form a real obstacle for creating computational platforms based on a database. The technological successes that were achieved for genome analysis have resulted in a set of notoriously large databases that are still actively developed:

- DDBJ/EMBL/GenBank contains sequences of nucleotides (genome sequences), which is basically the DNA code. This code is mapped to names and identifiers. Some parts within this code constitute the genes. The database contains also sequences of amino-acids, which are the building blocks of proteins. Three institutes, DDBJ [18] in Japan, EMBL [19] in Europe and GenBank [20] in the USA, have cooperated on this work since 1992. All together, in April 2012 it featured approximately 412 billion nucleotide base pairs in approximately 242 million sequence records, from more than 260 thousand different named organism groups.

- UniProtKB [21] is a KB about proteins that consists of two sections: Swiss-Prot, which is manually annotated and reviewed, and TrEMBLE, which contains DNA sequences, but lacks protein annotations that are reviewed. The latter section is much larger than the first, but the quality of the data is much poorer. The UniProt Consortium was launched in December 2003 as a merger of older resources of the European Bioinformatics Institute (EBI), the Swiss Institute of Bioinformatics (SIB), and the Protein Information Resource (PIR).

- PROSITE [22], created in 1988, records protein domains, functional sites and protein families. The knowledge in this KB is about entities that are smaller than entire proteins, but larger than the letters in the sequence code. Protein families are groups of evolutionarily related proteins.

- Entrez Gene [23] is a database about genes that were mostly computationally discovered from the analysis of genome sequences. The genes were discovered in the sequences of the RefSeq database [24]. New resources have been added since 2003.

- GeneID [25] is a database about genes that have been well investigated and it provides extra information about these genes. If focuses only on species whose genome was sequenced entirely and that have an active research community, like the genome of the fruit fly (Drosophila). GeneID version 1 was released in July 2000.

- KEGG Pathway [26], initiated in 1995, consists of networks of protein interactions in the cell, including also variants that are specific to particular species.

- IntAct [27], first published in 2004, is a database for molecular and protein interactions. The knowledge stems entirely from published literature and is manually annotated by expert biologists. It contains about 126 thousand binary interactions from 2100 scientific publications.

- Reactome [28], developed since 2003, is an open source, manually curated database about biological pathways. This includes information about proteins, like where and how they participate in the pathway. Reactome contains cross-references to many other databases, including different others in this list.

Large knowledge resources outside the domain of genome biology typically have a longer history. SNOMED CT [29], the Systematized Nomenclature of Medicine – Clinical Terms, is a medical terminology that should help physicians and health-care professionals in their communication about medical practices. It has evolved from SNOMED RT (Reference Terminology) and the United Kingdom's Clinical Terms Version 3 in a collaboration between the College of American Pathologists (CAP) and the National Health Service (NHS) in England. SNOMED itself has evolved from SNOP (Systematized Nomenclature of Pathology), which was started in 1965. Since 2007 SNOMED CT was adopted by the International Health Terminology Standards Organization (IHTSDO). The knowledge base contains over 311 000 unique concepts and approximately 1 360 000 semantic relations between the concepts (October 2011). Each concept is linked to

various terms or phrases, called Descriptions. These terms clarify the meaning of the concepts for humans or they might be used in medical texts to refer to the concept.

The standardization of medical ontologies has been particularly challenging. Knowledge management in health care is of great practical importance compared to other domains in the Life Sciences. The exchange of medical data between hospitals and health care practitioners has led to several standards for so-called information models. Taxonomies like the International Classification of Diseases (ICD) are designed for enabling efficient lookup by physicians. Medical ontologies try to create a comprehensive and consistent representation from a scientific viewpoint, while maintaining interoperability with standards that were designed for the medical practice.

The Internet, the World Wide Web and search engines, have revolutionized the field of Knowledge Management. These technologies have facilitated a strong tendency to integrate dispersed knowledge resources into a single, large distributed system. Relational database tables have not always helped in achieving this integration. Instead, they often appeal to rigid models that cannot be integrated with other rigid models. The search for overarching solutions has directed the attention of knowledge engineers towards Ontology, the philosophical field that investigates the description of everything. In recent years, most knowledge resources are released as so-called 'ontologies'. Many of them also commit to an unambiguous logical semantics, for example, since 2011, SNOMED CT is formally restricted to the semantics of the logic EL++ [30]. The theory behind Ontology and Logic, and behind technologies like XML, OBOF, RDF, OWL and SPARQL, will be explained in detail in Chapter 3.

Probably the most famous example of an ontology in the life sciences is the Gene Ontology (GO) [31]. It consists of the terminology that is needed to describe gene products, and therefore also the genes that encode these proteins. GO provides functions that proteins can have, cell locations where proteins can do their job, and cellular processes in which proteins can be involved. Complexes of proteins that are common among many species are also in GO in the form of cellular locations. GO, being an ontology, integrates with a classical relational database, called the Gene Ontology Annotations (GOA) [32]. It is a database in which each row contains an annotation of a protein with a term in GO, for about 2000 species. GO and GOA have increasingly been used in the field of Bioinformatics during the last decade.

Other examples of knowledge resources that were developed as ontologies include FMA [33], the Foundational Model of Anatomy ontology for anatomical knowledge of the human body, CHEBI [34], an ontology about Chemical Entities of Biological Interest, GALEN [35], an ontology about medicine and NCBI Taxonomy [36], an ontology about organisms. By adhering to some principles of ontology engineering, these resources have a more commonly understood semantic foundation and they can be managed more easily with common tools.

In order to ensure that ontologies in the biomedical domain can interoperate, the Open Biomedical Ontologies (OBO) Foundry [37] was initiated. It collects ontologies that adhere to a common set of ontology design principles, like openness, identifiers, orthogonality of content, versioning and more. At least 106 different ontologies were developed as OBO ontologies in February 2012, however, only 6 of them have passed the internal review system of the OBO Foundry (GO, CHEBI, PATO, PR, XAO and ZFA). Also relations between biomedical entities were assembled and reviewed in OBO's Relationship Ontology [38].

The OBO ontologies have a common format, the OBOF format [39], which makes it easy to treat them generically in a single software system. The language is very human readable for being a KR language that is also computer readable. This means that specialized biomedical scientists, who are not acquainted with computer science formalisms, can easily contribute by turning their knowledge into OBOF files. The drawback is that programmers need to do a lot of work to do any advanced KM tasks with these files. Currently, there is a standardized translation to OWL, called OboInOwl [40], which is of great use for programmers. However, any new developments to OBOF remain a heavy burden for software implementers.

Many other computer languages were developed for the representations of biomedical knowledge resources. SBML, the Systems Biology Markup Language [41] is a KR language, based on XML, for the representation of biological processes, like metabolic networks, cell-signaling pathways and regulatory networks. Such processes are stored as mathematical models. Also CellML [42] is an XML-based language that stores mathematical models, which are abundant in cell biology. A language that has tried to coordinate with different standards for representing biological pathway data is BioPAX [43]. The use of such languages have facilitated the modeling of very diverse biological systems in a common format that can be validated and managed by software tools.

2.5 Integrated Knowledge Bases

Modern databases and ontologies are engineered to be interoperable, however, they are maintained and released separately by groups of scientific experts. Other knowledge engineers take care of the task to integrate these separate resources into larger systems, called Knowledge Bases (KB). They should help scientists to retrieve the knowledge in their domain and to address the knowledge resources with broad queries. A semantic technology is a tool, a language or a standard that supports such KBs.

One of the most useful and best maintained integrated KBs is the NCBO BioPortal [44]. It provides a nice website for browsing any biomedical ontologies that are open and the system is relatively flexible for submitting new ontologies to it. Apart from search facilities for particular terms and ontologies, it also provides a graphical browsing interface. It is also possible to annotate texts with specific terms that are within any of the ontologies. A similar system for OBO ontologies is the Ontology Lookup Service (OLS) [45], which is a spin-off of the PRIDE project (PRoteomics IDEntifications database). PRIDE is a centralized, public data repository for proteomics data, which requires a query interface for ontology and controlled vocabulary lookup. OLS uses a relational database back-end for capturing OBO's Term and Typedef stanzas. Whereas NCBO BioPortal and OLS excel in user friendliness, Ontobee [46], another service for browsing OBO ontologies, provides good support for semantic technologies like SPARQL through a PHP enabled web interface. This gives users the chance to create more advanced queries, starting from the basic browsing queries that are provided.

Bio2RDF and BioGateway are two RDF KBs that have embraced semantic technologies for integrating a large number of different knowledge resources. Bio2RDF [47] presents itself as a platform for biomedical knowledge discovery. It integrates this knowledge by a syntactic and a semantic normalization in the Semantic Web language RDF. It allows for queries across the whole platform. BioGateway [7] is an integrated Knowledge Base that integrates all the OBO ontologies, Gene Ontology Annotations for about 2000 organisms, NCBI Taxonomy and Uniprot/SwissProt. The scope and the basic approach is similar as for Bio2RDF, although BioGateway facilitates the knowledge discovery process by providing many prepared queries. Moreover, by the use of Metarel [5], computational reasoning was achieved in BioGateway, which provides more useful answers to the user. Metarel and BioGateway are explained in chapters 5 and 6.

All the KBs described above are created by scientists for other scientists. However, Google [48, 49] and Wikipedia [50, 51] are so popular, also for scientific lookup, that they need no introduction. An interesting derivation of Wikipedia for genome researchers is Proteopedia [52]. It is an online encyclopedia that consists of knowledge about proteins and other molecules. Every page contains a continuously rotating 3D image of the protein and the whole site is very user friendly. Such websites are very promising, because the technologies on which they are built are much better supported than more specialized technologies. Eventually all the successful efforts on KM in the Life Sciences will become accessible through such kind of user-friendly websites.

2.6 Knowledge in biomedical literature

Apart from classical relational databases and ontologies, there is obviously a third type of knowledge resource: the scientific publication. They are expressed in what we call 'natural language', which is a very flexible format for expressing knowledge. Unfortunately, the challenges for integrating this knowledge become only larger. Millions of publications were made in the Life Sciences and this number has appeared to grow exponentially during the last decade [53]. Even biologists with a very narrow field of interest cannot read every paper that is of use for them. It is another task of KM to avoid that all this knowledge gets lost.

PubMed [54] collects biomedical literature from MEDLINE [55], Life Science journals and online books. This resource comprised more than 19 million citations and more than 2.2 million articles in July 2011. It is developed and maintained by the National Center for Biotechnology Information (NCBI). Many knowledge statements in biomedical KBs have a reference to literature through PubMed identifiers.

An important strategy for using the literature resources is automated text mining [56]. Software programs try to read and understand scientific literature in order to transform it to more structured formats that are easier to handle by other KM tools and search interfaces. Such an effort was executed by Van Landeghem et al., who used machine learning techniques to extract terms about genetic interactions from text [57]. Ontologies have proved to be useful resources for text-mining applications in biomedicine, by providing a semantic framework to extracted terminologies [58].

An example of an alternative search interface for PubMed is called

HubMed [59], which was described as the Swiss Army knife of PubMed interfaces. Genome experts are served with special search interfaces like iHOP [60], Chilibot [61] and BioMint [62], which provide special support for names and synonyms of genes and proteins. Whereas iHOP provides sentences with their context, Chilibot has extracted relations between proteins, genes and keywords from scientific literature into a giant graph. Just like BioMint, the interfaces have extracted their knowledge from PubMed, to which the results are linked. All these technologies make the extant literature much better accessible in automated ways.

Another strategy for the management of knowledge in literature is provided by Simplex [63], which is a system that converts literature input about regulatory relations between biomedical entities into differential equations. These can be imported into a mathematical programming environment for predicting the behavior of the regulatory system. Simplex expects manually translated input from literature and is not a text mining tool.

2.7 Open source and open data

Most software programs and the KBs mentioned in this thesis are open source. This means that everybody is allowed to see and understand all the internals of the system, to use it and to integrate it in other (open) software systems. The idea to build open systems is fundamental to computer science. Without open software architectures, programmers and knowledge engineers could not benefit from each other's efforts. They need each other's work as building blocks for the ultimate end-product. World-wide standards are very important to streamline the cooperation between programmers of open software. Both open software and world-wide standards play a key role for KM, just like for any discipline in computer science.

Chapter 3

Logic and semantics

Because of the rapid growth of biomedical knowledge during the last decade, a lot of recent research in Knowledge Management relates to biology. Naturally, this has not always been the case. The current research on KM and semantics (meaning) builds on the age-old study of logic that starts even more than two thousand years ago with Aristotle [64]. Two domains of research that sustain modern KM have developed throughout the ages: Logic and Ontology. Both domains will be introduced in this chapter, followed by an introduction on the state of the art, being the Semantic Web. This will give the reader some acquaintance with logic languages, classes, instances, decidability, tractability, Description Logics, ontologies and finally with Semantic Web standards like SPARQL, RDF and OWL. All these terms are of importance for understanding the following chapters.

3.1 Logic in history

Modern logic starts in the nineteenth century with the publication of *The Mathematical Analysis of Logic* [65] of George Boole in 1847. This work provided a system that could express logical relations in the form of algebraic formulas. Boole introduced symbols for selecting objects (elective symbols) that became the predecessors of the modern truth functions.

The next advancement in the study of logic was the publication of *Begriffsschrift* [66] by Gottlob Frege in 1879. Frege proposed a concept writing (or concept notation) that included symbols for negation, implication, universal and existential quantification and identity. These symbols have eventually resulted in the modern variants that are very well known

to logicians and mathematicians today. However, the concepts in Frege's system have to be considered as naive sets that lead to a paradox that was discovered by Bertrand Russell in 1901 [67]. These naive sets can contain themselves as a member, and they also allow to define the set of all the sets that do not contain themselves. This is known as the Russell paradox: if this set (the set of all sets that do not contain themselves) contains itself, it is defined to not contain itself and vice versa.

The Russell paradox was not quickly resolved and led to many attempts to resolve it. A direct result was the development of the Zermelo-Fraenkel set theory (ZF), which is still considered today as the canonical foundation for mathematics [68]. Another major consequence was the development of the incompleteness theorems by Kurt Gödel in 1931. They prove that any effectively generated theory capable of expressing elementary arithmetic cannot be both consistent and complete [69].

With the formulation of the *Entscheidungsproblem* ('the Decision problem' in English) by David Hilbert in 1928, logicians started to concentrate on the decidability of logical and mathematical problems. Hilbert wondered whether it would be possible to create, in theory, an algorithm (or a machine) that could decide for any given mathematical or logical hypothesis whether it was true or not. Today, logicians speak about the decidability of formal languages. Any mathematical or logical problem can be rewritten as a language representation (a series of symbols), for which the related decision problem is the question whether the representation is valid in the formal language or not. This way of speaking holds at least for languages that serve as a syntax. When a language serves as a semantics, its representations can be considered as expressions for which the decision problem is whether they are true or false.

Alonzo Church [70] and Alan Turing [71] provided a negative answer to the Entscheidungsproblem in 1936 and 1937 respectively: there exist general logical problems for which it is not possible to construct an algorithm that can decide whether a given case of the logical problem is true or false. Turing also provided an important example: the halting problem. The halting problem decides for any given computer program (which consists of a general algorithm and a particular problem on which the algorithm runs) whether this program will either halt or run forever. Turing proved that it is not possible to construct a general algorithm to solve the halting problem. This implies that the halting problem is undecidable. We can extrapolate this conclusion for formal languages: some formal languages are

decidable, others are not. For undecidable languages, it is impossible to create a validation service.

Alan Turing made a much bigger achievement by providing the fundamental architecture of modern computers [72]. His A-machine (automatic machine, now called Turing machine) consisted of a tape with symbols, a reader that reads a symbol and a table of states that decide which actions to take, depending on the previous state and the symbol that is read. Possible actions are reading or writing on the tape.

The effective construction of such computers through electrical transistors has paved the way for the research to computational complexity and the tractability of computer programs [73]. The key question in this research field is no longer whether a computer program will eventually halt (the question of decidability), but whether it will halt in a reasonable amount of time and with a reasonable amount of computer memory (the question of tractability). Computational complexity was defined formally by considering the behavior of generic algorithms for very large cases (inputs) of a logical problem. The relation between the minimally required computing resources and the size of the input for inputs that go to infinity, is defined as the complexity of a problem. For example, if the relation between the number of required steps and the size of the input of a problem is a polynomial function, then we can say that this problem is solvable in polynomial time. All the polynomial-time problems belong to the complexity class P. Other complexity classes are L (linear time), NP (verifiable in polynomial time), PSPACE (requiring a polynomial amount of memory space), EXPTIME (exponential time) and EXPSPACE (exponential memory space). However, it is not proven yet that all of these classes are really different. It would be of great practical importance to know for sure that some problems for which the solution is verifiable in polynomial time, are not actually solvable in polynomial time. This question, the P versus NP problem, was introduced by Stephan Cook in 1971 [74].

3.2 Logic languages

3.2.1 Propositional logic

The notion of 'formal language' that complexity theorists currently use, stands quite some distance away from the formal languages that are used in Knowledge Management. It replaces what is informally understood as a 'mathematical problem', whereas a particular language representation

(called a word in complexity theory) replaces 'a particular case of the mathematical problem'. The interest of complexity theorists in languages reduces to questions like 'Is this particular language representation valid in this language?' and 'To which other representations can it be transformed?'

The usage of formal languages within the field of Knowledge Management is much broader. What it adds to the languages is *semantics* (or meaning). A very basic mechanism to add meaning is to consider a valid language representation as a statement that is *true*. Such a representation is what we will call a **knowledge statement**. It is also called 'predicate', 'sentence', 'formula' or 'proposition'. It forms the basis of the *propositional logic*, also called *zeroth-order predicate logic*, which allows to infer new propositions (theorems) from propositions that were defined to be true (axioms), through a set of inference rules.

Here is an example of propositional logic. From the propositions

- If it is dark, then Leo hunts.
- It is dark.

it follows:

- Leo hunts.

3.2.2 First Order Logic

The next idea that adds meaning is the correspondence between the symbols that are used in the language and objects that exist in the world that it describes. This world is called the *domain of discourse*. Since the objects are described by classifying them into *classes*, they can be viewed as *instances* (or examples) of such classes. *First-Order Logic (FOL)* distinguishes itself from propositional logic by the use of quantifiers, like 'for all', 'there exists' or 'for exactly three'. In FOL, the quantifiers range exclusively over the instances in the domain of discourse. In higher-order logic, quantifiers can also range over classes and predicates.

Here is an example of FOL. The class 'liger' is defined by quantifying over all elements x in the domain of discourse. From the two FOL statements:

- For every x it holds that x is a liger if and only if x is the child of a male lion and x is the child of a female tiger.

- Sinbad is a liger.

  it follows:

- Sinbad is the child of a male lion.
- Sinbad is the child of a female tiger.

FOL is already quite *expressive*. The expressivity of a language is a measure for the number of ideas that can be expressed in the language. Through its domain of discourse and its quantification over elements in this domain, FOL is clearly more expressive than propositional logic. There is, however, a severe drawback: FOL is not decidable [67]. It is called semi-decidable, because sometimes statements may follow from the axiomatic statements, but there may also exist statements for which it cannot be decided whether they follow from the axiomatic statements or not. The undecidability of FOL implies that it is possible to construct paradoxes in this language. A famous example of a paradox is the barber paradox, which contains a self-reference [75]. The paradox is about a city where all the men remain always shaven, in two possible ways: either by shaving themselves, either by going to the barber. The barber is required to shave exactly those men who do not shave themselves. We have the following two axiomatic statements in FOL:

- There exists a barber x.
- For every y it holds that x shaves y, if and only if y does not shave y.

  This can be expressed more formally as follows:

- $\exists x\ barber(x)$
- $\forall y(shaves(x, y) \Leftrightarrow \neg shaves(y, y))$

The paradox emerges by asking who shaves the barber. The truth of the statement "The barber shaves himself" (or $\exists x(barber(x) \wedge shaves(x, x))$) cannot be decided. This paradox is related to the Russell paradox, which defines the set of all the sets that do not contain themselves.

Another way to formulate the undecidability of FOL is to say that no inference procedure can be created for FOL that is both sound and complete. A sound inference procedure is correct for all the statements that are

inferred by the procedure, but other possible statements, which are not inferred, cannot be claimed to be evaluated by the procedure. An inference procedure that is both sound and complete can correctly evaluate the truth of every possible statement.

3.2.3 Early Knowledge Representation languages

In the early 1970s, Prolog was developed as a logic programming language [76]. Although it was initially aimed to process natural language, it was given a formal underpinning through Horn logic, for which sound and complete inference procedures can be built [77]. Horn logic is a subset of FOL in such a way that it becomes a decidable language. Just like FOL, Horn logic uses conjunction ($\wedge$, the logical AND), disjunction ($\vee$, the logical OR) and negation ($\neg$, the logical NOT), but in every disjunction of literals, only a single literal is not negated. This restriction decreases the expressivity compared to FOL, but it makes Horn logic decidable in polynomial time. As a result, Prolog was one of the first logic programming languages that could support real applications in computer science. It is still popular today.

The search for logic languages that were practically useful continued in the 1980s. KL-ONE was developed as a knowledge representation language that could give a logical foundation to semantic networks [78]. In addition to a network with labeled nodes and connecting relations (the basis of a semantic network), it also provides a structured inheritance system. For example, if every bird has wings, and every eagle is a bird, then 'eagle' inherits the property of having wings from 'bird'. The usage of logic for the representation of knowledge was a driving force to increase the expressivity of the language, however, KL-ONE and other knowledge representation systems developed in the 1980s, like K-REP, BACK and LOOM, also paid attention to the tractability of the algorithms for making the logical inferences [79, 80, 81].

These early KR-systems were given inference procedures that operated in polynomial time, which is considered as tractable in computer science (exponential time in contrast is intractable). However, more detailed research brought to light that some modest extensions, adding only very little expressivity, resulted in intractable procedures. Even worse, certain logical queries were proven to be undecidable in KL-ONE [82].

3.2.4 Description Logics

The search for expressive logic languages, that were decidable fragments of FOL and that could be given tractable inference procedures, led to the birth of Description Logics (DL) [83]. These form a family of languages for knowledge representation purposes. They all share a common basis, which is the relatively inexpressive DL language AL (Attributive language). The terminology used in DL is slightly different. DL uses *concepts* instead of classes, *roles* instead of relations and *individuals* instead of instances.

The fundamental building block in DL is the *axiom*. A terminological axiom is about concepts and states to which broader concept a concept belongs (concept inclusion, for instance 'African animal is an animal that lives in Africa'). An assertional axiom states to which concept an individual belongs (concept assertion, for instance 'Dolly is a sheep') or by which role two individuals are connected (role assertion, for instance 'Asia is located in the Northern hemisphere'). Role assertions always connect individuals. Axioms that involve two concepts and a role are also terminological axioms. Finally there are also axioms about roles and that state to which broader role a role belongs (role inclusion, for instance 'regulates is broader than negatively regulates'). Table 3.1 contains a list of symbols that can be used in DL axioms. Since all the DLs are decidable, it can be decided for any other statement whether it can be inferred from the axioms or not. If the DL is also tractable, this inference can happen in a reasonable amount of time.

Concepts are divided in primitive and non-primitive (or defined) concepts. For example, if 'animal' is a primitive concept, 'Africa' an individual and 'lives in' a role, then the non-primitive concept 'African animal' can be defined as any animal that lives in Africa. For non-primitive concepts there exists a set of necessary and sufficient conditions for individuals to belong to the concept.

An important DL is ALC (Attributive Concept Language with Complements), which was introduced by Manfred Schmidt-Schauß and Gert Smolka in 1991 [84]. It has been the basis for many other DLs. Franz Baader has reformulated the language in a well-known system that is often used currently [85]. Such a system can be used to delineate different logics, which can be studied and compared together. The system itself is not a logic and does not warrant decidability of the logics that are expressed with it. The logic ALC is expressed as follows:

Let N_C, N_R and N_O be sets of primitive concepts, primitive roles

$\top$	the top concept: the concept containing all the individuals
$\bot$	the bottom concept: a concept that contains no individuals
$\sqcap$	intersection or conjunction of concepts (and)
$\sqcup$	union or disjunction of concepts (or)
$\neg$	negation or complement of concepts (not)
$\forall$	universal restriction (for all)
$\exists$	existential restriction (there exists)
$\sqsubseteq$	Concept inclusion (is a)
$\equiv$	Concept equivalence
$\doteq$	Concept definition
:	Assertion of concept (is instance of) or role (is related to)

Table 3.1: A list of symbols that can be used in DL axioms, and their meaning.

and individuals, let R be a primitive role and let C and D be ALC concepts. Then the following are also ALC concepts:

- $\top$ (top is a concept)
- $\bot$ (bottom is a concept)
- Every $A \in N_C$ (all atomic concepts are concepts)
- $C \sqcap D$ (the intersection of two concepts is a concept)
- $C \sqcup D$ (the union of two concepts is a concept)
- $\neg\, C$ (the complement of a concept is a concept)
- $\forall\, R.C$ (the universal restriction of a concept by a role is a concept)
- $\exists\, R.C$ (the existential restriction of a concept by a role is a concept)

If, for example, human is an ALC concept, then non-human (the complement of human) is also an ALC concept. If 'has child' is a primitive role, then 'individual that has a child' (existential restriction) is an ALC concept. If also 'female' is an ALC concept, then 'individual that has only females as children' (universal restriction) will be an ALC concept as well. In this manner, by using the ALC concepts in (series of) axioms of the form

$C \sqsubseteq D$, called general concept inclusions (GCI), many advanced concepts can be defined.

More advanced DLs can be defined by including more advanced types of concept constructors compared to those used in ALC. Also role constructors and role axioms can be included for creating more expressive DLs. Number restriction puts a minimum and/or a maximum on the number of role assertions for a certain role. For example 'busy parent' can be defined as any person that 'has child' at least two. Qualified number restrictions counts the number of roles towards a certain other concept. If 'planet' is a concept and 'has part' is a role, then 'planetary system' can be defined as anything that has part at least two planets. Some DLs can define roles from primitive roles through role constructors, like the inverse, the union, the complement or the intersection of one or more primitive roles. The role chain constructor defines roles that always imply the existence of a chain of two primitive roles. For example, if 'assists' and 'controls engine' are roles, then 'is co-controller of engine' can be defined as the chain of these roles. Role axioms are axioms about roles that provide a necessary condition for a role, without necessarily providing conditions that are sufficient for defining the role. Examples of role axioms are transitivity and role inclusion. For example, the role 'is preceded by' includes the role 'is directly preceded by', because it always holds when the latter holds. The usefulness of the role chain constructor is greatly advanced by using it in combination with role inclusion, which is called complex role inclusion. For example, it can be expressed that 'is located in' includes the role that is the chain of 'is part of' and 'is located in'. This implies that 'is located in' holds whenever the chain of 'is part of' and 'is located in' exists, without implying that this chain holds whenever 'is located in' exists.

The investigation of the tractability of the different DLs, as well as the need for comparison of DLs with other logical systems, for instance the Entity Relationship (ER) model [86], has necessitated to distinguish DL axioms in assertional axioms or A-box assertions and terminological axioms or T-box assertions. Assertional axioms are either the assertion that an individual is an instance of a concept or that an individual is related to another individual by a role. Terminological axioms are GCIs, which are about concepts. Axioms that are purely about roles, like transitivity or inverses, are sometimes distinguished in a separate R-box. The merger of the A-box, the T-box and the R-box is called a Knowledge Base in DL parlance. The tractability of logical questions in DLs is typically investigated

with an empty, a small or a large A-box and/or T-box.

The comparison between DL and relational database systems holds when ER schemes are considered as a DL T-box, and the database contents are considered as a DL A-box. In this view, database entities correspond to DL individuals. This way of equating ER schemes with DL T-boxes may be useful for comparing performance metrics. Conceptual modeling languages like ER have proven to hold a tight correspondence to DL-Lite [87]. However, as we will see in the next section, such conversions can lead to knowledge representations that are not ontologically sound. The database entities of biomedical databases are very often ontological classes that require a conversion to DL concepts in the T-box instead of individuals in the A-box. Biomedical ontologies that are translated to DL often have an A-box that is empty or almost empty, as is shown by the lack of individuals in OBO ontologies. Relational database systems on the other hand, for which all the contents reside in an ER scheme instead of tables, are usually avoided.

3.3 Ontologies

Just like Logic, Ontology is a domain of study in philosophy that has its origins in ancient Greece. It is the second pillar that sustains modern KM. The word 'ontology' is derived from the Greek words *ontos*, which means 'being', and *logos*, which means 'word' or 'study'. Ontology is the study of everything that exists. This field was also established by Aristotle [88], who grouped entities into categories and hierarchies. The word 'ontology', however, was probably invented only some centuries ago. The earliest extant mention of the word stems from 1613 in the Lexicon philosophicum by Rudolf Gockel [89].

An ontology, in its original meaning, was considered as a description of everything. In more recent times, say the last two decades, the meaning has evolved by the use of ontologies for knowledge management in computer science. An ontology is now considered as a description of a scientific domain. Consequently, there are many ontologies under construction. In theory, by taking them all together, they might be considered as the description of everything.

An often used definition in computer science is that of T. Gruber in 1993 [90]: an ontology is a formal, explicit specification of a shared conceptualization. However, this definition also includes computer scientific

representations for abstract and virtual domains. Such ontologies can even describe computer programs and terms that are used within the realm of the computer program. BFO and the OBO Foundry use a more classical definition by representing only the things that exist in reality [91]. BFO explicitly excludes the description of all sorts of conceptualizations, like ideas, representations and non-existing entities like unicorns.

Modern ontologies are typically organized in hierarchies that have very generic terms at the top levels. For so-called upper-level ontologies, these can be terms like 'thing', 'object', 'process', 'continuant' or 'function'. Ontologies about a more specific domain will define their scope by their most generic term in the hierarchy, also called the root of the ontology. An ontology about proteins will have 'protein' as root, an ontology about living organisms (taxonomy) may use 'organism', etc. Below the root come terms that are less generic, like 'p53 protein' and 'human p53 protein' in a protein ontology, or like 'mammal' and 'whale' in an ontology about organisms.

Every system or framework that represents knowledge needs to make similar choices concerning its relation to reality and its usage of very generic terms. The set of these choices is called the *ontological commitment* of the system. Systems with a very different ontological commitment are usually not *interoperable*.

3.3.1 Instances and classes

An important distinction for ontologies is the distinction between particulars (also called individuals) and universals [92, 93]. These terms have often been debated throughout the history of philosophy. However, they can easily be equated with instances and classes in logics. The distinction between these two is not anything more mysterious than the distinction between proper names and nouns. In English, a proper name is written with a capital whereas a noun starts with a lower case. Therefore we write 'Jupiter', but not 'Planet'.

The distinction between instances and classes is an important ontological commitment that is universally accepted in the field of Ontology. However, it is often misused by knowledge engineers who are building practical applications. For instance, in an ontology about cars, they may be tempted to model the Mercedes car (the class of all cars from the trademark Mercedes) as an instance by giving it the name 'Mercedes' and by claiming that it represents the (individual) trademark, and not a class of

cars. This stands in contrast with the principle of representing the things that exist in reality, which implies that knowledge about this domain should be expressed as knowledge about the cars instead of the trademark. Every class can be logically represented as an instance in this way. Problems will occur if ontologies with arbitrary distinctions between instances and classes are integrated into a larger knowledge base. For example the question 'how many cars are ecological in Belgium?' in an integrated knowledge base would reveal the problem. An ontology that had chosen trademarks as instances might return a number lower than ten, which appeals to an arbitrary classification of cars into types of cars. An ontology with the cars as instances might return a number of thousands or larger.

Similar confusions of speech happen in the domain of biology. When a biological expert claims that his research is concentrated around 'only three proteins', he will surely refer to three classes (or types) of proteins. Even more specifically, he will refer to three types of proteins that are produced from an identical (or very similar) genetic code, rather than three types that are folded in the same way. It may be good to remember that, in ontologies, two (instances of) physical objects, like two proteins, are only considered 'identical' or 'the same', if they share exactly the same three-dimensional spatial coordinates, as well as the same time coordinates. In other words, they must be *one*. It is not sufficient to look indistinguishably similar.

Individual proteins are so small and there are so many of them, that they will never get proper names (not speaking of science fiction worlds). This fact may be very surprising for logicians, database engineers and object-oriented programmers, who are used to work only with instances that can be referred to with identifiers. This is why research on biological ontologies seems often quite alienated from research on ontologies that is inspired by practical applications in the domain of logics. The absence of any identified instances in ontologies about microscopic entities calls for other applications.

3.3.2 Relations in ontologies

There are different formalisms to represent the modern ontologies in computer science. The most important formalisms are logics from the family of Description Logics. But almost every modern ontology is represented as a collection of terms (classes and instances) and relations between these terms.

There are different types of relations. The most important relations are the subsumption relations that constitute the classification hierarchy. The relation between a class and a subclass is called *is subclass of* or *is a*. The relation between an instance and a class is called *is instance of*. All taken together these relations form a graph that cannot have cycles. In other words, the subsumption relations form a directed, acyclic graph (DAG).

The subsumption relations do not necessarily use *single inheritance*, for which every class can have only a single superclass, forming a tree-shaped hierarchy. In general, ontologies support *multiple inheritance*. In that case the 'Indian elephant' can be a subclass of both 'elephant' and 'Asian animal'.

Other important relations that form a DAG are the partonomic relations, which are called *is part of*. Whereas a subsumption hierarchy would descend from bird to owl to screech owl, a partonomy will descend from bird to bird body to wing to feather. Every screech owl *is a* bird, which can not be said for feathers. Every feather *is part of* a bird.

Both the partonomic relations and the subsumption relations are transitive. This means that if A *is a* B and B *is a* C, then also A *is a* C. These relations are also reflexive. This means that every A *is a* A, and also every A *is part of* A. From this fact it follows that the subsumption hierarchy is always a subset of the partonomic hierarchy. If A *is a* B, then also A *is part of* B. This amounts to saying that every owl is part of a bird, simply because every owl is part of itself.

There are many other types of relations that are used between terms in ontologies. They are called *is located in*, *is preceded by*, *activates*, *regulates*, etc. Some of these are transitive, some are subrelations of others (for instance every activates relation is a type of regulates relation), and some even form chains (for instance if A *is located in* B and B *is part of* C, then A *is located in* C).

3.3.3 An example: the Cell Cycle Ontology

Ontologies can generally be subdivided in domain ontologies and application ontologies. Domain ontologies describe a certain scientific domain. The most famous example is the Gene Ontology, describing the domain of genome biology.

Application ontologies describe more specific domains by reusing domain ontologies. A good example of an application ontology in the biomedical domain that is developed with modern techniques is the Cell

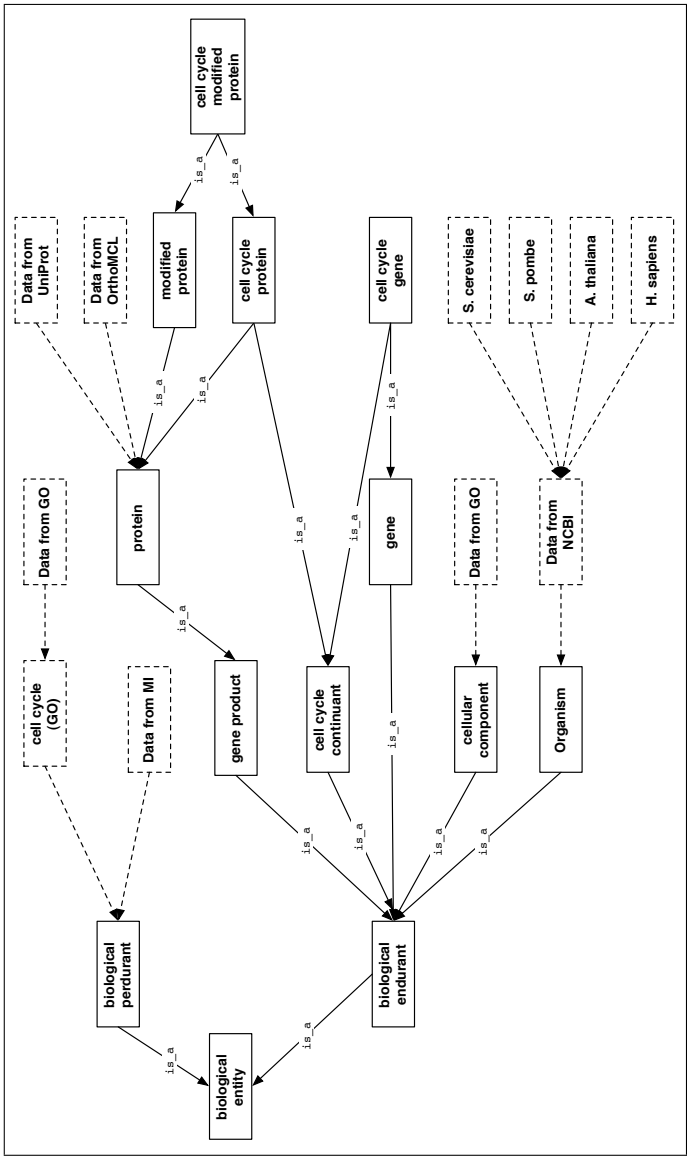


Figure 3.1: The upper level classes of CCO. The dashed boxes represent external data sources, whereas the continuous boxes are ontological classes that were specifically minted to integrate the external data sources in CCO.

Cycle Ontology (CCO) [2], which was developed at VIB/UGent and is discussed in more detail in Chapter 4. This ontology integrates recent knowledge about the cell cycle, the process of duplicating a biological cell into two daughter cells. The cell cycle consists of a series of subprocesses and hundreds of genes are involved in the entire process. Detailed knowledge about these genes, like their interactions, their functions and other processes they involve in, are spread over different knowledge bases. Scientists who are interested in the cell cycle can use CCO to find all the concepts that are related to this process, as well as the relations between these concepts.

The most generic class in CCO (the root) is called 'biological entity'. This means that the instances in CCO are all the biological entities that exist in the universe. In practice, not a single instance in CCO is explicitly identified by a unique identifier or a name. The knowledge about the cell cycle is entirely represented by classes and relations between classes. The instances of biological entity form the domain of discourse, but they are not addressed individually.

For the management of relations, CCO has integrated RO from the OBO foundry, and has reverted to the inclusion of Biorel later, which will be introduced in Chapter 6. Some definitions were initially provided by the developers of CCO itself, like 'catalyses', 'codes for', 'degradates' and interacts with for biomolecules and 'has source' for organisms. Synonyms were also provided, like 'catabolises into' and 'decomposes into' as exact synonyms of 'degradates'.

Figure 3.1 shows the subclasses to the right of the root. They are often represented underneath the root, with the leaves (the most detailed classes) on the bottom. CCO makes a distinction between perdurants (time-dimensional processes) and endurants (3D objects) that are derived from formal ontology. Further subclasses, like gene product and organism, are more comprehensible for biological domain experts. All the links indicated with *is_a* can be understood as a formal, logical subclass relation. Multiple inheritance is used for some classes, for instance for 'cell cycle modified protein', which is a subclass of both 'modified protein' and 'cell cycle protein'. The logical implication is that 'cell cycle modified protein' is a subclass of the intersection of both superclasses.

CCO uses the OBO format, however, it is not orthogonal with other OBO ontologies. Since CCO is an application ontology, it reuses classes from domain ontologies. For instance 'cell cycle' and 'cellular component' stem from the Gene Ontology.

3.4 The Semantic Web

Many technologies in computer science have evolved very rapidly since the beginning of the nineties, when more and more computers got connected world-wide to form the Internet (or the web). The cooperation of HTML [94, 95], a language for linked computer documents, and HTTP [96], a protocol for transferring computer documents, gave birth to the World Wide Web. This was established by Tim Berners-Lee in 1990 [97]. The World Wide Web can be considered as the human readable part of the Internet.

In 1994, the World Wide Web Consortium (W3C) was founded for designing new standards for the World Wide Web [98]. Tim Berners-Lee, who became director of the W3C, set ambitious goals for the further development of any related technologies. He envisioned the Semantic Web as an extension of the World Wide Web [99]. This extension would make documents and data on the Internet also understandable for computers. The Semantic Web and its technologies facilitate advanced queries and applications that can reuse and exchange data that was given a meaning.

The Semantic Web technologies are built on the same basis as the World Wide Web. It is currently viewed as a stack of technologies: the famous Semantic Web Stack [100] (see Figure 3.2). The bottom of the stack consists of Unicode [101] for characters and IRIs [102] for identifiers. Above that there is a syntax layer of XML [103]. This in turn supports RDF [104, 105], a layer for graph-based representations that facilitates intuitive queries with SPARQL [106]. RDF supports more expressive representation languages like RDFS [107] and OWL [108, 109]. The logic-based sublanguages of OWL (OWL profiles) give a logical meaning to data and they facilitate reasoning [110]. By the encryption of the data, Internet users can have some trust in the Semantic Web machinery.

3.4.1 Unicode

The standard for characters used in Semantic Web technologies is the Unicode character set [101, 111], which consists of millions of symbols, with an upper limit of more than 1 billion symbols (see Figure 3.3). Chinese characters, Arabic script and script symbols from any other language have their unique place in the Unicode character set. The standard for identifying the correct symbol by a series of bits within a binary text file is given by the 8-bit Unicode Transformation Format (UTF8) [112]. The architec-

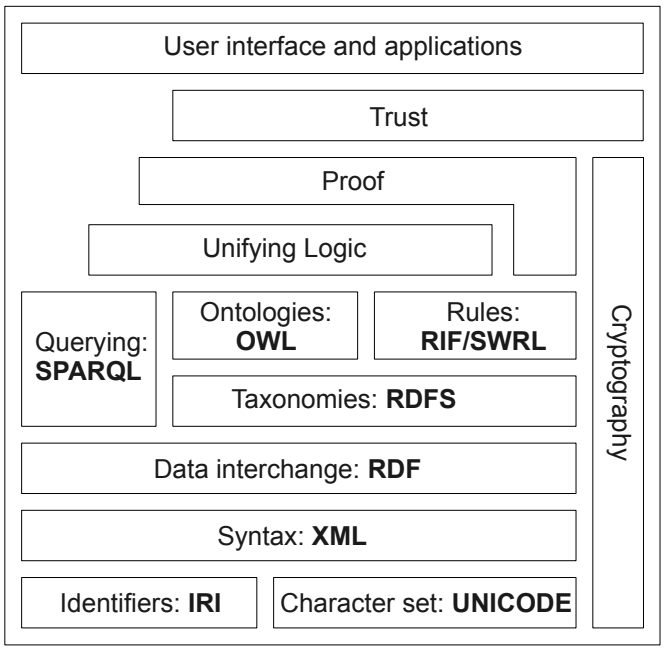


Figure 3.2: The Semantic Web Stack has to be viewed from bottom to top. The more advanced technologies (RDF and OWL) are represented on top, as they are sustained by the technologies on the bottom (IRI and Unicode).

ture of UTF8 is compatible with its predecessors, the ASCII and ASCII-II tables [113], which identified only 128 and 256 symbols respectively.

3.4.2 IRIs

The use of global identifiers, that can be shared world-wide among information systems connected to the Internet, form a very important component of the Semantic Web. Identifiers are for computers what words in a natural language are for humans. If two information systems use different identifiers, they will not be able to communicate successfully, just like people who are speaking a different language.

The W3C has adopted a generic syntax for identifiers as a compact series of characters that identify abstract or physical resources. Such identifiers were initially called Uniform Resource Identifiers (URIs) [114]. They

0020	0 0030	@ 0040	P 0050	` 0060	p 0070	00A0	° 00B0	À 00C0	Ð 00D0	à 00E0	ð 00F0
! 0021	1 0031	A 0041	Q 0051	a 0061	q 0071	¡ 00A1	± 00B1	Á 00C1	Ñ 00D1	á 00E1	ñ 00F1
" 0022	2 0032	B 0042	R 0052	b 0062	r 0072	¢ 00A2	² 00B2	Â 00C2	Ò 00D2	â 00E2	ò 00F2
# 0023	3 0033	C 0043	S 0053	c 0063	s 0073	£ 00A3	³ 00B3	Ã 00C3	Ó 00D3	ã 00E3	ó 00F3
$ 0024	4 0034	D 0044	T 0054	d 0064	t 0074	¤ 00A4	´ 00B4	Ä 00C4	Ô 00D4	ä 00E4	ô 00F4
% 0025	5 0035	E 0045	U 0055	e 0065	u 0075	¥ 00A5	µ 00B5	Å 00C5	Õ 00D5	å 00E5	õ 00F5
& 0026	6 0036	F 0046	V 0056	f 0066	v 0076	¦ 00A6	¶ 00B6	Æ 00C6	Ö 00D6	æ 00E6	ö 00F6
' 0027	7 0037	G 0047	W 0057	g 0067	w 0077	§ 00A7	· 00B7	Ç 00C7	× 00D7	ç 00E7	÷ 00F7
(0028	8 0038	H 0048	X 0058	h 0068	x 0078	¨ 00A8	¸ 00B8	È 00C8	Ø 00D8	è 00E8	ø 00F8
) 0029	9 0039	I 0049	Y 0059	i 0069	y 0079	© 00A9	¹ 00B9	É 00C9	Ù 00D9	é 00E9	ù 00F9
* 002A	: 003A	J 004A	Z 005A	j 006A	z 007A	ª 00AA	º 00BA	Ê 00CA	Ú 00DA	ê 00EA	ú 00FA
+ 002B	; 003B	K 004B	[005B	k 006B	{ 007B	« 00AB	» 00BB	Ë 00CB	Û 00DB	ë 00EB	û 00FB
, 002C	< 003C	L 004C	\ 005C	l 006C	| 007C	¬ 00AC	¼ 00BC	Ì 00CC	Ü 00DC	ì 00EC	ü 00FC
- 002D	= 003D	M 004D	] 005D	m 006D	} 007D	- 00AD	½ 00BD	Í 00CD	Ý 00DD	í 00ED	ý 00FD
. 002E	> 003E	N 004E	^ 005E	n 006E	~ 007E	® 00AE	¾ 00BE	Î 00CE	Þ 00DE	î 00EE	þ 00FE
/ 002F	? 003F	O 004F	_ 005F	o 006F	007F	¯ 00AF	¿ 00BF	Ï 00CF	ß 00DF	ï 00EF	ÿ 00FF

Figure 3.3: The 0020-00FF range in the Unicode character set, also known as the Latin Alphabet No 1 character set. The table shows the graphic character symbol on the left and the hexadecimal character value on the right for each character record. This range is just a tiny part of the whole range of the Unicode character set, which has reserved space for more than a billion characters.

can identify anything: websites, documents, data, but also things in the real world like people, buildings, countries, the Earth and the universe. All these things are called *resources*.

URIs are more generic than Uniform Resource Locators (URLs), which identify a unique location on the Internet [115]. They are also more generic than the Uniform Resource Names (URNs), which are names for resources that are not necessarily located on the Internet (see Figure 3.4). Special schemes exist for URLs and URNs, in such a way that URIs have an identifier space that includes the locator space and the name space that are specified by the schemes.

In 2005, the Internationalized Resource Identifier (IRI) was specified as the recommended generalization of the URI [102]. The only difference with the URI is the allowed set of characters. For IRIs, every Unicode char-

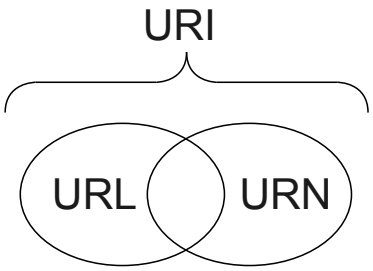


Figure 3.4: Uniform Resource Identifiers (URIs) as a superset of Uniform Resource Locators (URLs) and Uniform Resource Names (URNs).

acter can be used, which includes also characters from Chinese, Japanese, Russian and many other languages. We can notice that every technology in computer science requires a certain compromise between the understandability for computers and the understandability for human users and developers.

3.4.3 SGML

The Standard Generalized Markup Language (SGML) [116, 117] is a very generic computer readable language that is used by HTML and XML. The textual symbols that are meant to give the computer readable structure to the text, are called ‘markup’. For example, the tags surrounding the following title "<title><boldface> On the origin of species. </boldface></title>", are markup and they indicate that this is a title and it should be represented in boldface. SGML is useful as a syntax for any structured format, because it separates the symbols that are used for structuring from the symbols of the normal text or data. This separation is the basis for any automated analysis of text by a computer, which is called ‘parsing’.

3.4.4 XML

The Extensible Markup Language (XML) is a sublanguage of SGML. XML was engineered to facilitate extensions to the set of syntax rules. By adding more restrictive syntax rules, many sublanguages of XML can be easily created. The validity of a representation as XML can be checked with a single validator (like the W3C Markup Validation Service) whereas the validation

of an XML sublanguage will require some extensions to the existing software for validating XML. As such, XML is perfectly suited to serve as a syntax layer for the Semantic Web. Several other Semantic Web languages were given an XML syntax, which means they are sublanguages of XML.

Technically, XML has the structure of a tree, with a root, branches and leaves, or even a collection of many trees. XML can be queried with XPath [118] and XQuery [119].

3.4.5 HTML

The most important technology for representing documents on the Internet is the Hypertext Markup Language (HTML). The technology is a part of the World Wide Web and it is used to represent human readable documents, which are transferred by the HyperText Transfer Protocol (HTTP). In that sense HTML is not really a part of the Semantic Web, however, it builds on the same standards. The location of every document on the Internet is indicated with a URL, and therefore also with an IRI, which is the basis for the Semantic Web.

HTML was developed in 1991 by Tim Berners-Lee as a sublanguage of the more generic markup language SGML. It was intended to make scientific text at the CERN institute more accessible. The most important feature is the linking of different documents through hyperlinks. Every hyperlink is presented to the user as a clickable word that hides the URL of another document. Berners-Lee also created the first browser for HTML documents, called WorldWideWeb [120]. More features were added to the HTML language when Microsoft and Netscape started to develop their own browsers.

Being a markup language, HTML can also represent boldface, italics, fonts, tables, figures and much more. The W3C has taken care of new recommendations for HTML, which has evolved rapidly during the nineties. The last recommendation, HTML 4.01, was released in 1999 [121] and was followed by the publication of an ISO/IEC international standard called 'ISO HTML' in May 2000.

ISO HTML is a sublanguage of SGML, but it is not a sublanguage of the more restrictive XML. This way of developing HTML has ended in favor of XHTML 1.0 [122], which became a W3C recommendation in January 2000. XHTML has remodeled HTML syntactically as a sublanguage of XML. Between 2000 and 2009, any updates to HTML were all released as later versions of XHTML.

Since September 2009, a new standard called HTML5 has been under development. The final release of this standard is severely hampered by the pending status for standardization of many related Web technologies, like audio and video.

3.4.6 RDF

The Resource Description Framework (RDF) is a Semantic Web language that can store resources and connections between resources as a large set of so-called *triples*. RDF/XML [123] is the preferred syntax for RDF, which is both valid RDF and valid XML.

Every RDF triple consists of a subject, a predicate and an object. The subject is a resource (represented with an IRI) or a blank node (represented by a local identifier), the predicate is always a resource and the object is either a resource, a blank node or a literal (any series of Unicode characters).

RDF can be queried with SPARQL (SPARQL Protocol And RDF Query Language), which has a non-verbose, intuitive syntax. Queries are submitted to a SPARQL endpoint, which is often accessible worldwide on the internet. The SPARQL query language is quite similar to SQL [124], the query language for relational databases. It became an official W3C recommendation on 15 January 2008. The worst-case complexity analysis of the SPARQL language predicts that queries require a polynomial amount of memory space (PSPACE) in function of a combined measure of the length of the query and the size of the RDF store [125]. This is harder than any NP-complete problem and in the worst case the computation will require an exponential amount of time in function of this combined measure.

Six months later, on 15 July 2008, SPARQL Update (SPARUL) [126] was submitted to W3C. SPARUL is an extension of SPARQL for making modifications (updates) to RDF in a store, instead of only querying RDF. This tool appears to be very powerful, as will be shown in this thesis. It is even possible to facilitate semi-automated reasoning with SPARUL, even though the tool was created for making atomic updates in RDF stores. In the beginning of 2012, SPARUL was in the last review stages for becoming a W3C recommendation.

Other syntaxes for RDF exist as well, like the RDF/TURTLE syntax [127] and the RDF/N3 syntax [128], which are much less verbose compared to the standard RDF/XML. The Turtle (Terse RDF Triple Language) syntax for RDF was accepted as a first working draft by the W3C Work-

ing Group in August 2011. The syntax is a subset of the more general N3 (Notation 3) syntax for RDF. Both Turtle and N3 are much more human readable and easier to use compared to the recommended RDF/XML syntax, which has better options for advanced programming environments.

Many RDF management systems are built on top of relational database management systems. RDF stores with SPARQL are currently a very promising candidate for large-scale KM systems that are compatible with the principles of the Semantic Web.

3.4.7 RDFS

RDF is very useful as a flexible, technical language for validation, querying and making updates, however, it does not give much clues for a universally understood logical meaning of its representations. It gives some basic clues by the indication that the first part of the triple is the subject, that the second is the predicate and that the third is the object. This makes a clear correspondence to simple natural language sentences with a subject, a verb and an object. More advanced logics are required by the Semantic Web community, which led to the development of the Resource Description Framework Schema (RDFS). RDFS became a W3C recommendation in 2004.

RDFS consists of a vocabulary for concepts in modern logic, like instances, classes, subclasses, properties and subproperties. It also includes 'resource', a central concept of the Semantic Web. Table 3.2 shows the vocabulary for classes of RDF resources like literals, properties and datatypes. Table 3.3 shows the vocabulary for properties, like 'is subclass of' and 'see also', which are used for relating RDF resources.

Unfortunately, there is no unambiguous method to construct a valid RDFS representation. The RDFS vocabulary does not overcome the problems and paradoxes that Bertrand Russell encountered more than a century ago. RDFS is not a decidable logic which means that different implementations of validator services and reasoners may come to different conclusions. Whereas several good RDF validator services are available on the Internet, nothing could be built for RDFS because of this reason. However, the vocabulary is used abundantly in other Semantic Web standards like RDF and OWL.

Class name	comment
rdfs:Resource	The class resource, everything.
rdfs:Literal	The class of literal values, e.g. textual strings and integers.
rdf:XMLLiteral	The class of XML literals values.
rdfs:Class	The class of classes.
rdf:Property	The class of RDF properties.
rdfs:Datatype	The class of RDF datatypes.
rdf:Statement	The class of RDF statements.
rdf:Bag	The class of unordered containers.
rdf:Seq	The class of ordered containers.
rdf:Alt	The class of containers of alternatives.
rdfs:Container	The class of RDF containers.
rdfs:ContainerMembershipProperty	The class of container membership properties, rdf:_1, rdf:_2, ..., all of which are subproperties of 'member'.
rdf:List	The class of RDF lists.

Table 3.2: RDF vocabulary for classes.

Property name	comment	D.	R.
rdf:type	The subject is an instance of a class.	R	C
rdfs:subClassOf	The subject is a subclass of a class.	C	C
rdfs:subPropertyOf	The subject is a subproperty of a property.	P	P
rdfs:domain	A domain of the subject property.	P	C
rdfs:range	A range of the subject property.	P	C
rdfs:label	A human-readable name for the subject.	R	Lr
rdfs:comment	A description of the subject resource.	R	Lr
rdfs:member	A member of the subject resource.	R	R
rdf:first	The first item in the subject RDF list.	Ls	R
rdf:rest	The rest of the subject RDF list after the first item.	Ls	Ls
rdfs:seeAlso	Further information about the subject resource.	R	R
rdfs:isDefinedBy	The definition of the subject resource.	R	R
rdf:value	Idiomatic property used for structured values (see the RDF Primer for an example of its usage).	R	R
rdf:subject	The subject of the subject RDF statement.	S	R
rdf:predicate	The predicate of the subject RDF statement.	S	R
rdf:object	The object of the subject RDF statement.	S	R

Table 3.3: RDF vocabulary for properties. Domains (D.) and ranges (R.) are either rdfs:Resource (R), rdfs:Class (C), rdf:Property (P), rdfs:Literal (Lr), rdf:List (Ls) or rdf:Statement (S).

3.4.8 OWL

The Web Ontology Language (OWL) is a technology that stands at the top of the Semantic Web stack. It can serve for representation of knowledge through a logic-based meaning. Its development has resulted in 19 different W3C recommendations, ranging from the treatment of datatypes, to the usage of syntaxes and OWL profiles.

This language is constructed from several earlier languages, like OIL and DAML [129, 130], and it is compatible with other standards, like XML and RDF. The computer understandable meaning of OWL is guaranteed by the research in Description Logics. The first release of OWL in 2004 distinguished three sublanguages: OWL Full, OWL DL and OWL Lite. It turned out quickly that OWL Full suffered from the same problem as RDFS. Since it is not based on a decidable logic language, it is not even possible to create a validator for OWL Full. The language is considered as the superlanguage that contains all the language constructs, which means that OWL Full can serve as a vocabulary that extends the vocabulary of RDFS. OWL DL was based on Description Logics from the very start [131]. It is still the most expressive logic language that was created for the Semantic Web, however, computational reasoning in OWL DL may not always be efficient. OWL Lite was developed as a logic-based sublanguage of OWL DL. Every OWL Lite representation is therefore also a valid OWL DL representation. Consequently, OWL Lite is less expressive compared to OWL DL. The advantage of using OWL Lite should lay in a better performance for reasoning and queries. OWL Lite is optimal for very specific applications that require reasoning and querying about identified instances. Such applications lay mostly outside biomedical knowledge management.

In 2009, OWL was upgraded with a new release to OWL 2. OWL 2 has several sublanguages, called OWL profiles [110]. An important profile is OWL 2 DL, the successor of OWL DL, which is a superlanguage of several other OWL profiles. Three new profiles were added: OWL 2 EL, OWL 2 RL and OWL 2 QL. Especially OWL 2 EL appears promising for biomedical knowledge management, since it is a sublanguage of OWL 2 DL that facilitates polynomial time reasoning on large data sets [132].

Since the introduction of OWL 2, the language constructs within OWL, RDF and RDFS can be used in two different semantical contexts. The first is the Direct Semantics (DS) [133], which suits DL-based profiles of OWL, like OWL DL and OWL EL. This semantics assumes a proper distinction of instances and classes. The second semantical context is given

by the RDF-Based Semantics (RBS) [134], which is suited for data that is represented in OWL Full or RDF(S). However, the answer to some logical questions within RBS cannot be decided unambiguously.

The recommended standard for the syntax of OWL is OWL/RDF/XML. This syntax builds further on the RDF/XML syntax, which means that every OWL file in this syntax can be loaded in any RDF store. In theory, it is possible to query these OWL files with SPARQL (because it is valid RDF) and even with XPath (because it is valid XML). In practice it will be very hard to exploit all the logical expressivity of OWL through these lower query languages. An OWL profile like OWL DL requires a reasoner for querying. Reasoners that were built for Description Logic languages, like Pellet [135] and Kaon2 [136], were easily adapted to read the OWL/RDF/XML syntax. Other reasoners, like SPARQL-DL [137], were developed specifically for OWL DL and its recommended syntax.

In spite of all the research in all the efforts that are made to create different OWL profiles and their corresponding reasoners, it remains a great challenge to create queries for large-scaled OWL applications. First of all, the engineers of OWL queries need to agree upon the same logical model as the engineers of the OWL knowledge base. This is often very difficult to achieve, since the rigidity of logics requires many modeling choices that are not intuitive to understand. Secondly, the hardness of computational reasoning is a severe obstacle. OWL DL reasoners require exponential time for solving the worst cases and even reasoners for OWL profiles that warrant polynomial time may respond too slowly.

3.4.9 Rule languages

The Semantic Web Rule Language (SWRL) [138] was developed in 2004 as a combination of the rule language RuleML [139] and OWL. Rules, which specify logical inferences, can be used to extend the expressivity of a logic with the risk of loosing decidability. More specifically, SWRL extends the expressivity of OWL's decidable sublanguages like OWL DL and OWL Lite. SWRL has not become a W3C recommendation, unlike the Rule Interchange Format (RIF) [140], which was endorsed by the W3C in 2010. RIF expresses rules in XML and defines a system that can import RDF and OWL. Unfortunately RIF is not intuitive for developers who are acquainted with OWL or RDF and its development was focused on many other use cases. Moreover, RIF is partially redundant in scope with OWL profiles, which can also make logical inferences. SPARQL/Update pro-

vides an alternative for declaring rules in RDF, which will be shown in this thesis. The SPARQL Inferencing Notation (SPIN), submitted to W3C in 2011 [141], may complement this approach by providing an RDF syntax for SPARQL-expressed rules.

3.4.10 Linked Data

The paradigm of Linked Data extends the basic principles of the Semantic Web with some extra rules. The rules were not officially endorsed by the W3C, but they can facilitate more integration and accessibility of knowledge and data. Tim Berners-Lee, director of the W3C, has expressed the ideas in a technical note in 2006 [142]:

- Use IRIs as names for things.
- Use HTTP IRIs so that people can look up those names.
- When someone looks up an IRI, provide useful information, using the standards (RDF, SPARQL).
- Include links to other IRIs, so that they can discover more things.

The most important difference with the endorsed Semantic Web principles is the idea that every IRI should be *dereferenceable*. This means that any common web browser (e.g. Firefox) can retrieve information about the IRI, just like for the URLs of human readable websites. By the endorsed Semantic Web principles, IRIs are used for identifying resources in RDF stores and querying with SPARQL, but they are not necessarily supported by common web browsers.

Linked Data is RDF data that is *browsable*. One of the reasons why RDF is often hard to browse with SPARQL is the presence of many blank nodes that connect nodes identified by IRIs. Tim Berners-Lee defined a graph G as browsable if the lookup of any IRI node in G returns information which describes the node, where 'describing a node' is defined recursively as

1. returning all statements where the node is a subject or object; and
2. describing all blank nodes that are attached to the node by one arc.

It will certainly cost more investment in software systems to make IRIs accessible both in SPARQL endpoints and for general web browsers. Developers of RDF stores are currently working on this.

In 2010, still in the same personal note, the director of the W3C added a star rating system as a road to good linked data, particularly for government data. Good Linked Data:

$*$	is available on the web (whatever format), with an open license.
$**$	is available as machine-readable structured data (e.g. Excel instead of an image scan of a table).
$***$	uses a non-proprietary format (e.g. CSV instead of Excel).
$****$	uses open standards from W3C (RDF and SPARQL) to identify things, so that people can point at it.
$*****$	links the data to other people's data to provide context.

In spite of the unofficial manner of publishing and standardizing the paradigm of Linked Data, the idea has become well known in the Semantic Web community. It reflects the authority of Tim Berners-Lee, the architect of both the World Wide Web and the Semantic Web. A journal publication in 2009 has affirmed the success of the ideas [143].

3.5 SPARQL tutorial

3.5.1 RDF graphs with triples

SPARQL has become one of the most popular Semantic Web tools because of its intuitive syntax for querying that makes it easy to learn. The queries specify graph patterns by a set of triples, where some places of the triples are taken by variables.

The following represents a fictitious RDF graph of four triples:
a b c.
a d e.
b d f.
c d g.

where
a is http://www.semantic-systems-biology.org/SSB#P35568

b is http://www.semantic-systems-biology.org/SSB#has_function
c is http://www.semantic-systems-biology.org/SSB#GO_0005515
d is http://www.w3.org/2000/01/rdf-schema#label
e is "Insulin receptor substrate 1"
f is "has function"
g is "protein binding"

a, b, c and d are IRI resources, whereas e, f and g are called literals. The four triples form a typical RDF representation of the intuitive knowledge statement "Insulin receptor substrate 1 *has function* protein binding". The first triple (a b c.) serves for automated reasoning and knowledge integration, the other three are used for attaching a human readable label to the IRIs. The triples can be represented more efficiently by using prefixes and the Turtle syntax:

```
@prefix ssb:<http://www.semantic-systems-biology.org/SSB#>.
@prefix rdfs:<http://www.w3.org/2000/01/rdf-schema#>.

ssb:P35568 ssb:has_function ssb:GO_0005515.
ssb:P35568 rdfs:label "Insulin receptor substrate 1".
ssb:has_function rdfs:label "has function".
ssb:GO_0005515 rdfs:label "protein binding".
```

Let us now take a look at how we can query this small RDF representation through the query language SPARQL. The following SPARQL query defines a tiny graph pattern that can match (or *bind*) the triples in the RDF representation:

```
PREFIX ssb:<http://www.semantic-systems-biology.org/SSB#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
SELECT ?x ?y ?z
WHERE {
  ?x ssb:has_function ?y.
  ?x rdfs:label ?z.
}
```

The WHERE-clause of the SPARQL query consists of a graph pattern with two triples. The triples are separated with a point, which represents the logical 'AND'. They might also be separated by the UNION operator, which would represent the logical 'OR'. There is exactly one possible binding of this graph pattern on the graph of four triples represented above. The pattern binds the following two triples:

a b c.
a d e.
Variable x binds a, variable y binds c and variable z binds e. The SPARQL query will return the following answer:

x	y	z
http://www.semantic-systems-biology.org/SSB#P35568	http://www.semantic-systems-biology.org/SSB#GO_0005515	Insulin receptor substrate 1

In theory, there is no restriction on the number of variables in the SPARQL queries. Many queries will ask only for a single variable, whereas more advanced queries may consist of ten or even more variables. In practice, such advanced queries may become quickly unresponsive. Also the number of answers may be much more than the single result in this example. A graph pattern consisting of one triple with three variables (?x ?y ?z), will bind all the triples in the RDF store exactly once, returning the whole RDF store as an answer. However, many RDF data providers have put an upper limit on the number of answers that will be returned to avoid an overload of the server.

3.5.2 RDF stores with quads

RDF stores are sometimes called *triple stores* by some, and *quad stores* by others. Technically, RDF stores consist indeed of quads rather than triples. This is because the triples are divided over RDF graphs in the store. Each of the graphs has its own IRI, just like the subject, predicates and objects in the triples. Therefore RDF stores contain quads that consist of four identifiers: a subject, a predicate, an object and a graph. An RDF store cannot contain two quads with exactly the same four identifiers. Just like every triple is unique within an RDF graph, every quad is unique within an RDF store.

The following represents a fictitious RDF store with four quads:
a b c h.
a d e i.
b d f i.
c d g i.

where
a is http://www.semantic-systems-biology.org/SSB#P35568
b is http://www.semantic-systems-biology.org/SSB#has_function

c is http://www.semantic-systems-biology.org/SSB#GO_0005515
d is http://www.w3.org/2000/01/rdf-schema#label
e is "Insulin receptor substrate 1"
f is "has function"
g is "protein binding"
h is http://www.semantic-systems-biology.org/SSB
i is http://www.semantic-systems-biology.org/labels

If we want to write the same query on this RDF store as compared to the query on the RDF graph in the previous section, we need to address the two graphs in the store explicitly. SPARQL has a convenient syntax for this:

```
PREFIX ssb_base:<http://www.semantic-systems-biology.org/>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX ssb:<http://www.semantic-systems-biology.org/SSB#>
SELECT *
WHERE {
 GRAPH ssb_base:SSB {
  ?x ssb:has_function ?y.
 }
 GRAPH ssb_base:labels {
  ?x rdfs:label ?z.
 }
}
```

This query will return exactly the same answer sheet with one result. The separation of the fourth term as an RDF graph is not only apparent in the syntax SPARQL, but even much more so in the system of loading RDF files into a store and exporting them from the store. The endorsed RDF/XML syntax for RDF files contains only triples. When an RDF file is loaded into an RDF store, the triples are always directed into a single RDF graph. Once they are in the store, it is convenient to redistribute them to other graphs with SPARQL/Update queries.

3.5.3 An RDF federation with quintets?

On the 7th of June 2011, the W3C published SPARQL 1.1 Federated Query working draft [144], an extension to the SPARQL 1.1 Query specification. Federated queries are SPARQL queries that are sent to different RDF stores (also called RDF services or RDF endpoints). Answers are retrieved from all the addressed stores and combined to produce a final answer. Since also

these RDF services are identified with IRIs, we can already envision *RDF quintets* as a way to identify RDF knowledge statements uniquely in an *RDF federation* (or simply the Semantic Web). Such a quintet will consist of a subject, a predicate, an object, a graph and a service.

Let us consider the following four fictitious quintets on the Semantic Web:
a b c h k.
a d e i k.
b d f j l.
c d g i k.

where
a is http://www.semantic-systems-biology.org/SSB#P35568
b is http://www.semantic-systems-biology.org/SSB#has_function
c is http://www.semantic-systems-biology.org/SSB#GO_0005515
d is http://www.w3.org/2000/01/rdf-schema#label
e is "Insulin receptor substrate 1"
f is "has function"
g is "protein binding"
h is http://www.semantic-systems-biology.org/SSB
i is http://www.semantic-systems-biology.org/labels
j is http://www.metarel.org/biorel
k is http://www.semantic-systems-biology.org/biogateway/endpoint
l is http://www.metarel.org/sparql_endpoint

In this case, k and l should be SPARQL services (which is not actually the case for l). If a SPARQL query engineer knows that service l contains labels for biomedical relations, he could create the following query:

```
PREFIX bgw:<http://www.semantic-systems-biology.org/biogate
way>
PREFIX metarel:<http://www.metarel.org/>
PREFIX ssb_base:<http://www.semantic-systems-biology.org/>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX ssb:<http://www.semantic-systems-biology.org/SSB#>
SELECT ?subject ?predicate ?object
WHERE {
 SERVICE bgw:endpoint {
  GRAPH ssb_base:SSB {
   ?s ssb:has_function ?o.
```

```
  }
  GRAPH ssb_base:labels {
   ?s rdfs:label ?subject.
   ?s rdfs:label ?object.
  }
 }
 SERVICE metarel:sparql_endpoint {
  GRAPH metarel:biorel {
   ssb:has_function rdfs:label ?predicate.
  }
 }
}
```

Using the short notation with letters, the graph pattern that is defined in the SPARQL query might be represented as follows:
?s b ?o h k.
?s d ?subject i k.
?s d ?object i k.
b d ?predicate j l.

A federated query engine is supposed to evaluate all the possible substitutions in the variables (preceded with the ?-symbol), and all their combinations, that exist on the whole Semantic Web. If both services respond well, the following should be among the results:

subject	predicate	object
Insulin receptor substrate 1	has function	protein binding

Federated querying cannot be hard for such a tiny example. The problem becomes much worse for billions of triples, distributed over several RDF stores. The optimization of SPARQL query execution within a single RDF store is already a major challenge and many queries remain unresponsive or are even immediately rejected by the SPARQL engine. The optimization of federated queries, for which large numbers of partial results may have to be sent back and forth over the globe, will naturally be much more cumbersome.

3.6 Personal outlook

Expressive DLs have gained the top position of the Semantic Web in the form of OWL profiles. They can solve certain logical questions in polyno-

mial time, which has been identified in computer science as a sweet spot for computational problem solving. However, it is important not to overestimate the value of polynomial time reasoning. Without any doubt, computational tasks like optimizing the profile of airplane wings and the fitting of many transistors on a small CPU are served by polynomial time solutions. Such computations lead to final results, which are valuable if their parameters are optimized above a certain threshold.

In KM, not all computational tasks are served by polynomial time solutions. The response time of queries for browsing and lookup should not depend on the size of the data that is not of use for the submitter of the query. Instead, the response time has to fit within the limits of his patience, or he will look for another system that is better targeted towards his personal needs. This means that queries for lookup should answer within some seconds, which means in constant time. The building of consistent ontologies, on the other hand, is very well served by polynomial time reasoners. Ontology engineers need to check the soundness of their ontological frameworks through advanced reasoners that operate on small modules. They can also create logical inferences during a heavy computational session, and store the results for lookup later.

The Semantic Web supports this view on KM very well by providing RDF for querying and OWL for building consistent ontologies. Technologies for RDF, like federated querying and SPARUL rules, are currently being explored. They can be used to modularize RDF stores in such a way that queries are launched over relevant data and that heavily queried modules are computationally better supported. Efforts to create better standards and more convergence for RDF are currently investigated and will be important to reach this goal [145].

The Semantic Web is much less suited for mathematical computations and for the management of unprocessed data. Relational database systems like MySQL [146] and mathematical packages like Matlab [147], should be preferred for these tasks. Pattern recognition in Semantic Web data may only be useful for linguists, since all the data is processed and organized by humans beforehand.

Chapter 4

Querying biomedical knowledge representations

4.1 Introduction

Biologists, the end users of biomedical knowledge bases, are not skilled in querying knowledge with computer languages. They feel most attracted to graphical user interfaces for the querying of knowledge. Instead of looking into numbers, statistical p-values and name codes in spreadsheets, they prefer to see colorful graphs. The knowledge in a good KR system is captured by the mind and integrates with ideas, as opposed to the original resources, which may look like a meaningless matrix. In this chapter, we will investigate the querying of biomedical knowledge through SPARQL, with an emphasis on queries that are used for visualization and browsing.

Many different querying systems and visualization systems have evolved in the Life Sciences to help the scientists in digesting the knowledge in very different areas of the biomedical domain. Biomedical scientists are confronted with e.g. 3D representations of molecules, abstract cell pictures, graphical sequence representations, biological networks and clustered arrays of expression data. They have often developed a strong sense of graphical thinking. Knowledge engineers should meet their needs by presenting new knowledge visually, thus enabling it to integrate with other visual representations that are already present in the mind of the biologist.

Designing a query system for visualization is a step that is preceded by a series of knowledge management operations, like annotation, knowledge acquisition, knowledge integration and reasoning. During this process

different types of metadata were added to the knowledge: universal identifiers, cross-references, comments, citation information and statistical measures. A good knowledge representation will not confront the biomedical scientist with a raw dump of this mixture of knowledge and data. The challenge consists of using the metadata appropriately for filtering and coloring just those parts of the knowledge that are useful for the end user.

This tasks constitutes a severe challenge for large integrated systems with knowledge derived from different areas. In fact there is not a single graphical view, nor any single query, that suits every need. A good KR system will force tool builders to provide special support for specific areas. The tool builder may even be required to learn in-depth about several aspects of the knowledge in that area. In spite of these issues, the field of Knowledge Management should also investigate generic methods for querying and visualizing knowledge. It is the only manageable approach for presenting large integrated knowledge bases like RDF stores to users.

We will investigate the possibilities of querying and visualizing RDF by exploiting the semantics of RDF relations. We will derive the theory by going to the practice with the use of the RDF query language SPARQL. The scientific domain of the cell cycle, and the Cell Cycle Ontology (CCO) in particular, will serve as a use case. This investigation will lead to the creation of a library of SPARQL queries that are suited for visualizing ontologies represented in RDF. The library, initially developed for visualization, will evolve into a more generic library for tackling standard query problems in RDF. This will eventually pave the way for the emergence of BioGateway, an RDF store with an RDF data model that was tested and optimized with this library.

4.2 Standard queries for browsing and visualization

Some requests for visual displays are independent of the scientific area of interest because they are translations of questions that are universal. We will distinguish three universal questions that can apply to any term in a scientific domain:

- What is this term?
- Which other terms are related to this term?
- Are there any more specific examples for this term?

The first question, 'What is this term?', asks for superclasses of the term in an ontology. These are also called 'parents' for direct superclasses and 'ancestors' for superclasses in general. For instance, if a researcher wants to know what 'ubiquitin-protein ligase inhibitor activity' means, he may learn from the direct parental term that it is a 'ubiquitin-protein ligase regulator activity'. This is however not a very informative answer. For this reason it may be a better idea to show all the ancestral terms. This will teach him that ubiquitin-protein ligase inhibitor activity is an 'enzyme regulator activity' and a 'molecular function'. The collection of all the ancestral terms of any ontology term is called **the path to the root**. The root in a tree-shaped hierarchy of an ontology will typically be a very generic term like 'entity' or 'thing'.

Another way of answering the first question is by providing a definition, synonyms and cross-references to similar or equivalent terms. The researcher may learn that ubiquitin-protein ligase inhibitor activity "stops, prevents or reduces the activity of a ubiquitin-protein ligase".

This term, the ubiquitin-protein ligase regulator activity, denotes a function. It is obviously related to the term 'ubiquitin-protein ligase', which is not a function but a cellular component. It will therefore not show up as an ancestor of the function, nor will the function be an ancestor of the cellular component. The term 'ubiquitin-protein ligase' should be given as one of the answers to the second question: 'Which other terms are related to this term?' This standard query is also called **get the neighborhood** of the term.

The third question, 'Are there any more specific examples for this term?', asks for either subclasses or instances of the term. This question will be most useful for very generic and abstract terms. Similarly as for superclasses, direct subclasses are also called 'children' whereas subclasses in general are called 'descendants'. The visual representation of all the descendants is usually not a good idea because there will be too many for generic terms. **Get the children** on the other hand is a standard query in a browsing system that starts with showing only the most generic root term to the user. Clicking on that term launches a query to get the children of the root term.

These three basic queries can also be used for the automated generation of graphical representations. This constitutes an extra challenge that will however not be treated in this thesis. The automated generation of figures requires mature software that can interpret and filter the result set

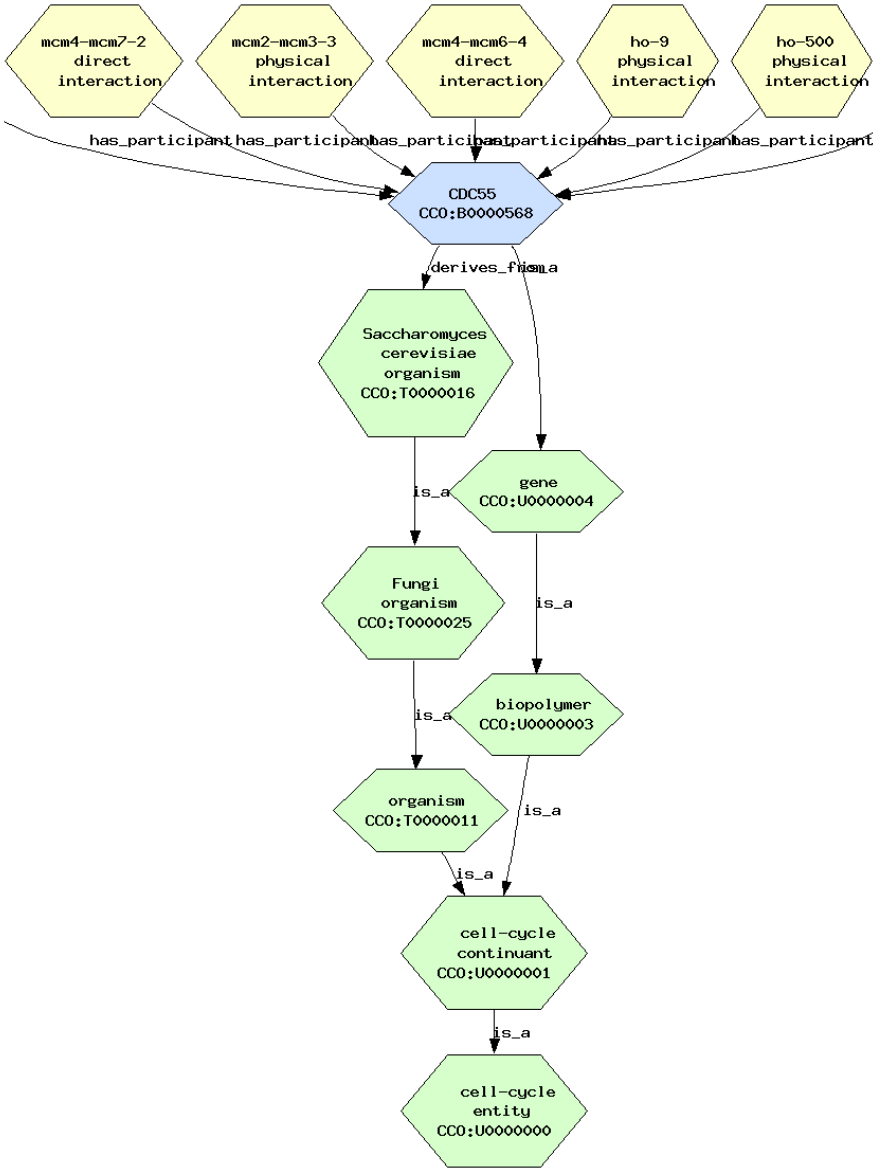


Figure 4.1: The neighborhood (top) and the path to the root (bottom) for the term CDC55 in CCO, generated with the Ontology Lookup Service software. The automated generation of figures that give a clear representation of results on a generic query is a challenge. Results on biomedical queries are often too abundant and names of entities are too long for fitting in a clear overview. Some labels are overprinted many times here.

according to its size. Figure 4.1 shows some of the problems that may be caused by automatically visualized queries. The figure was visualized with a local rebuilt of the software of OLS [45], which is based on DOT [148]. Another solution was finally used for the DIAMONDS platform, which is introduced in the next Section.

4.3 Querying the Cell Cycle Ontology

4.3.1 The DIAMONDS platform

The Cell Cycle Ontology [2] is an application ontology that integrates detailed knowledge about the cell cycle, a key process in biology that is of interest to many genome researchers. CCO was first published in 2006 [149] as a computational platform that was developed for the DIAMONDS project, an EU sixth framework project (FP6) for the life sciences, genomics and biotechnology for health [150]. The ontology aimed to enable the study of the cell-cycle process in four well-known model organisms: two yeasts, *Saccharomyces cerevisiae* and *Schizosaccharomyces pombe*, the plant *Arabidopsis thaliana* and finally human (or *Homo sapiens*).

ONTO-PERL [1], an Application Programming Interface (API), was specifically developed for creating and updating CCO. ONTO-PERL can also be used to create other ontologies in the OBO format. An automated pipeline coded in PERL uses modules in ONTO-PERL for recreating CCO. The pipeline was designed with the ambitious idea to recreate CCO fully automated overnight, including the downloading of original sources from the Web. However, the practice has showed that new downloads often include bugs that cause errors in the modules of ONTO-PERL. Many hands-on work has been done for the creation of ONTO-PERL and for each recreation of CCO.

Several data sources containing cell-cycle related knowledge are merged into CCO. The cell-cycle branch of the Gene Ontology [31] provides a hierarchical framework for the conceptual knowledge in CCO; Gene Ontology Annotations [32] contain relations about locations, processes and functions between cell-cycle related proteins and this hierarchy; protein-protein interactions come from IntAct [27] and UniProt [21]; whereas extra information about the four model species is obtained through adding the relevant branches of NCBI Taxonomy [36]. The core format of CCO is the OBO format [39], which is subsequently used for making exports to other

formats like OWL-DL [110], RDF [104], GML [151] and DOT [148]. The OWL export has been used for investigating logical consistency and new ontology design patterns [152].

The standard queries for visualization – the path to the root, the neighborhood and the children – only correspond to three interesting questions that can be visualized relatively straightforward. There is, however, a myriad of queries that are potentially interesting for biologists who are querying a knowledge base. Such queries can involve more than one or two terms and relations. Biologists are interested in advanced and complex relations between entities like proteins, processes, molecules, environmental conditions, diseases, species and many more. The visual representation of the results of such queries, and *a fortiori* the implementation of a graphical interface for making such queries, is a much harder challenge.

The final goal of the DIAMONDS project was to implement a graphical interface for generic queries about the cell cycle: the DIAMONDS platform, an EU Deliverable (D5.4). The Cell Cycle Ontology was successfully delivered as a part of the DIAMONDS project in the form of several valid ontology formats. Noray Bioinformatics S.L. (NorayBio), a commercial company, was the responsible partner within the EU project for the implementation of the DIAMONDS platform. Apart from the tabs for the visualization and navigation of CCO, the DIAMONDS platform contained also a section for the analysis of gene expression profiles.

The OWL-DL format of CCO was investigated in the research group of Robert Stevens in Manchester in 2007. The original idea, as defined in the DIAMONDS project, was that OWL-DL would be the preferred format for launching queries and for enabling any computational or logical operations. Initial small versions in the OWL-DL format could be handled, however, operations on larger versions turned out to be hopelessly slow. Even well-equipped servers could not load the whole CCO in OWL-DL. The RDF format on the other hand gave much better results. CCO could be loaded in RDF and queried with SPARQL, the RDF query language.

For this reason OWL reasoners were abandoned and replaced by Virtuoso [153], an RDF database management system, for handling the RDF format of CCO. In this way, a SPARQL endpoint provides access to CCO for any graphical query front-end. The endpoint can literally be considered as a point where the internal computational calculation of queries inside Virtuoso ends and where results are passed on, either to end-users, or to higher level software like the graphical interface built by NorayBio. The

endpoint provides explicit support for both possibilities: an HTML-format of the query results for human users and an XML-format for the processing by other software.

In this stage of the development, the following desiderata were established for visualizing CCO in the DIAMONDS platform:

- Terms should be represented by colored, clickable nodes with a name inside.
- The colors should reflect the type of the term, like protein, gene or pathway.
- Extra information about a term, like the definition, synonyms and comments should be given in a side-panel.
- Clicking on a node should return the local neighborhood.
- There should be an option to return the path to the root.
- It should be possible to select multiple nodes to retrieve their closest connecting path of relations.

The last desideratum was the most problematic one and there were different ways of defining the exact question. However, it was the only visualization feature that did not correspond with a standard query. Moreover, the idea of a closest connecting path holds the promise of computing and visualizing biologically relevant paths within the protein interaction network inside CCO. Because of these reasons this desideratum was considered as experimental and it was investigated separately.

The visualization applet of the DIAMONDS platform was finally published as part of CCO's publication in Genome Biology [2]. A screenshot is shown in Figure 4.2.

4.3.2 Integration through IRIs in a single identifier space

CCO was designed to exist in the form of five different ontology files, one for each of the four model organisms (cco_A_thaliana, cco_S_cerevisiae, cco_S_pombe and cco_H_sapiens) and one integrated file that included all the other files (cco). In addition, the integrated CCO would contain orthology data (this indicates which proteins in the four model organisms were

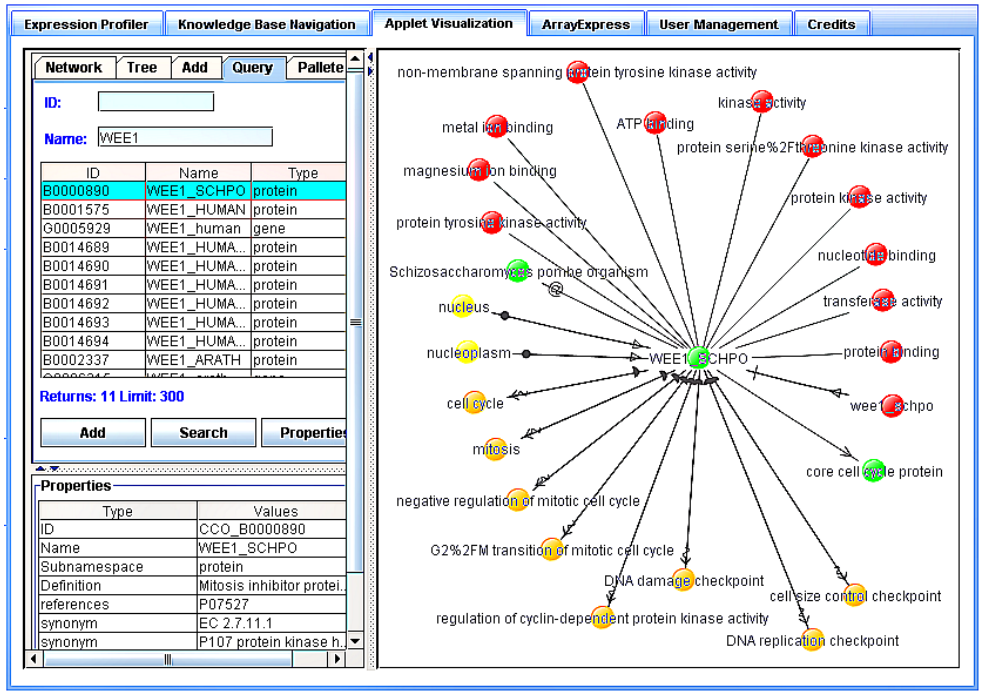


Figure 4.2: The knowledge visualization applet in the DIAMONDS platform. A query was launched to search the terms that contain WEE1 in the name. WEE1_SCHPO, a class of proteins in the *Schizosaccharomyces pombe* organism, was selected by a mouse click, which launched two other queries to retrieve the local neighborhood (visualized on the right) and the properties (bottom left) of this term.

expected to be evolutionary descendants of the same ancestral protein hundreds of millions of years ago, when the organisms had not yet diverged into different species). The reason for keeping more than only the integrated file was partly the specific interest of biologists who were studying just one of these four model organisms. An obvious other reason was the gain in computational performance for operations on the smaller files. However, the RDF translations into five graphs for the different files caused a major problem for querying across the borders of the different graphs with SPARQL. This problem, and the solutions that were created for it, are explained here.

The bottom layer of the Semantic Web consists of the IRIs, which identify the resources used. The W3C recommends the use of HTTP-style IRIs. Initially, the following namespace was used for creating such IRIs:

`http://www.cellcycleontology.org`

Through the functioning of domain name servers, anybody with a web browser (like Firefox) that is connected to the Internet could verify that this namespace was not for sale anymore. Even if the IRI does not point to any website, this information can be retrieved from the error code that is returned to the web browser. It provided the opportunity to create any IRI, starting with http://www.cellcycleontology.org, for the resources in CCO.

The idea of IRIs is to use only a single IRI per resource. Ideally this principle of uniqueness applies for the whole world. This theory stands in contrast to the ease of maintenance of the identifiers in a separate identifier space that is under control of the software engineers for a certain knowledge base. This easy solution had been used on different levels for CCO, which caused the problems for querying the RDF translations.

Querying over CCO and its original sources

All the external resources that are integrated in CCO get a new identifier assigned that start with the prefix ‘CCO’. For instance

GO:0007049,

the official OBO identifier for the term ‘cell cycle’ in the Gene Ontology, turns into

CCO:P0000066

in the identifier space of CCO. This practice may cause problems for querying. A query system in a knowledge base that contains both these identifiers cannot address the term 'cell cycle' in a unique way. It is possible to launch a query within the part of the system that uses only the CCO identifiers, or the part that uses only the original OBO identifiers, but not across the borders of both identifier spaces. It may be very useful to make such queries, because CCO integrates only specific sections of the original sources.

It must be mentioned that CCO contains cross-references to the original identifiers. In this way there is a formal correspondence between the two identifier spaces that would – in theory – allow to make queries that range over both CCO and the original sources. From experimental investigations with SPARQL, this theory turned out to be infeasible in practice. Every resource that is addressed in a query would need to be addressed twice through the use of UNION operators and some extra AND requirements to follow the cross-reference. This will make the query hard to write, hard to understand and especially hard to compute for the query server. On the other hand, CCO was designed to allow useful queries within itself, without requiring the parts of the original sources that were not integrated. This situation called for the construction of SPARQL queries that were possible within CCO itself.

At the moment that this design decision was taken for CCO, cross-references were a recommended practice within the OBO community. This mentality is slowly changing and the OBO Foundry has adopted IRIs as identifiers. It is one of the future plans of the developers of CCO to abandon the CCO-specific identifiers and exclusively use the identifiers of the original sources.

Querying across the model organisms in CCO

Creating identifiers and identifiers spaces that are unique world-wide is an ideal that is not easily achieved. Querying and maintaining resources that integrate knowledge from several more original sources would become more difficult when many different namespaces are used. This issue has to be weighed against the problems that arise from cross-referencing to identical objects. However, the initial RDF exports of CCO had also relied too much on cross-referencing for queries within CCO.

Since CCO consisted of five different ontology files, and since these were also translated into five different RDF graphs, the challenge remained to make queries across the borders of different RDF graphs for CCO. This brings up a very similar problem as the one with the reassigned and cross-referencing GO identifiers in CCO. The five graphs were all given a different identifier space, with identifiers that include the name of the model organism for each of the term. For instance, for the term 'cell cycle' in the RDF graph for *Homo sapiens*:

`http://www.cellcycleontology.org/ontology/rdf/Hs#CCO_P0000066`

This made it impossible for SPARQL to consider the five graphs as one large graph that contained only a single term 'cell cycle'. However, the solution for this problem required only a minor change in the ONTO-PERL code for the translation from OBO to RDF. Provided with this feedback, CCO's developers created a single identifier space for the IRIs within CCO. This would result in identifiers with the following syntax for this term:

`http://www.cellcycleontology.org/ontology/rdf/CCO#CCO_P0000066`

This identifier space was an important requirement for enabling queries throughout the RDF graphs of CCO. It installs the bottom piece of the Semantic Web Stack. The reporting of some other issues and bugs have increased the quality of the RDF translation tool in ONTO-PERL. The attempts to query across the RDF sources of all the original sources would later be succeeded in BioGateway, which is explained in Chapter 6.

4.3.3 SPARQL queries for browsing requests

A good architecture for the identifiers within a namespace, as explained in the previous section, has enabled the development of SPARQL visualization queries for the DIAMONDS project. The SPARQL queries were included in the platform's Java code, which was developed by NorayBio. The visualization module was based on a Java subroutine that was written by Steven Vercruysse, who donated his code for this purpose. This subroutine did not create static figures, but it optimized the space between the nodes and the length of the arcs interactively. It enabled that users zoom in and out, delete nodes and add nodes with clicks or new queries. The problem of overwriting in a static display like Figure 4.1 are solved in this way, as can be seen in the figures 4.3 and 4.4.

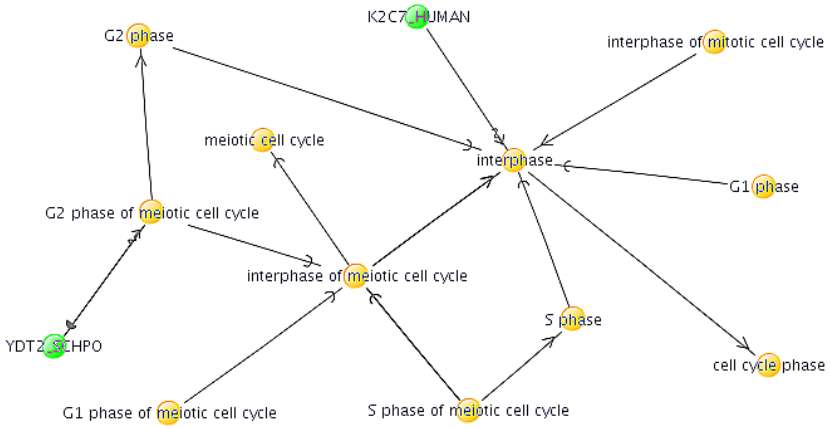


Figure 4.3: This screenshot shows some important cell cycle phases in CCO, together with two protein types. By a series of relevant queries and deletions of irrelevant nodes, users can select and visualize networks, pathways or protein interactions that are of interest to their particular research. The visualization is rendered by the Java subroutine that was written by Steven Vercruysse and integrated later in the DIAMONDS platform. The space between the nodes and the length of the arcs are optimized automatically by the subroutine, in order to avoid that labels and nodes overwrite each other.

Get the children: the basis for browsing

For launching generic types of visualization queries, like 'get the children of a certain term', it does not suffice to have a list of hard-coded SPARQL queries. The node that is clicked should appear as a parameter in the query. The results, the child terms, should be reusable as parameters in other queries. The IRIs in the unique identifier space that was created for CCO could perfectly serve as parameters. In fact it was possible to use only the part of the IRIs that appeared after the #-symbol as a parameter, because all the relevant, clickable nodes were identified within this identifier space for CCO. The following query was included in the DIAMONDS software for retrieving the children of any term:

```
# NAME: get_children
# PARAMETER: CCO_U0000000:
# The term for which the children will be found
# PARAMETER:
# http://www.cellcycleontology.org/ontology/rdf/CCO:
# The default ontology IRI
# FUNCTION: returns the children of the query-term

PREFIX default_ontology:
<http://www.cellcycleontology.org/ontology/rdf/CCO#>
PREFIX query_term_id:
<http://www.cellcycleontology.org/ontology/rdf/CCO#CCO_U0000
000>

SELECT ?child
FROM <http://www.cellcycleontology.org/ontology/rdf/CCO>
WHERE {
  ?child default_ontology:is_a query_term_id:.
}
```

Lines that start with the #-symbol are comment lines for SPARQL, so they could be used to indicate where the parameters occurred in the real code. The PREFIX-section serves to reduce long parts of IRIs, that are used several times, to a shorthand notation. Actually even entire IRIs can be replaced with a shorthand notation through the use of PREFIX. This is very useful for parameterizing SPARQL queries, because it allows to put all the Java-coded parameters in the PREFIX section, followed by a block of SPARQL code after the SELECT statement that was hardcoded in the Java software code. In the example above, the prefix *query_term_id* replaces the whole IRI

```
http://www.cellcycleontology.org/ontology/rdf/CCO#CCO_
U0000000
```

This is especially useful for queries where the parameter IRI occurs more than once in the SPARQL code after the SELECT statement. This practice of parameterizing through prefixes is even indispensable for an investigation of an RDF knowledge base with manually launched SPARQL queries. For a human who is copy-pasting IRIs, it is very impractical to search for all the occurrences of the IRI in the code and replace it everywhere.

The prefix *default_ontology* shows that this query deals with a poorly defined identifier space for the IRIs. The *is_a*-relation type, used for finding the subclasses of a CCO term here, is supposed to be a universal concept. Ideally, there should be only a single identifier in the whole world for it. This query assumes that it might be identified differently even in different RDF graphs within the same RDF knowledge base. The creation of Bio-Gateway later on would be the answer to this problem. In the meantime the fully integrated CCO ontology, comprising all the four model organisms, would serve as the 'default ontology' for querying.

The meaning and use of the query above is very basic. It finds all the children of a given term. This query is required in a browse system that presents the most generic term (the root of the ontology) to the user. By clicking on any term, the children will appear in the form of a list. Every term in the ontology can eventually be reached in this way.

Get the name and the type for visualization support

Each answer to a user query is visualized in the DIAMONDS platform. A system of colors is used to distinguish the type of terms that are returned, like for instance protein, gene, organism or biological process. For each term in the answer, the name is written over the nodes and the type is used to give the node a color. This means that the IRI of each term needs to be passed as a parameter in a query that retrieves the type and the human readable name of the term.

The graphical representation of the colored nodes can in turn be used to initiate new queries. Another option is the selection of specific nodes that are of interest, and delete the other nodes. A graphical representation of the most important terms that are related to the cell cycle, and the relations

between them, can be seen in Figure 4.3.

The following query provides the basis for the visualization support:

```
# NAME: get_name_and_type
# PARAMETER:
# http://www.cellcycleontology.org/ontology/rdf/CCO:
# The default ontology URI
# PARAMETER: CCO_B0000000:
# The term for which you want the name and the type
# FUNCTION: returns the name and the type for a given term

PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX query_term_id:
<http://www.cellcycleontology.org/ontology/rdf/CCO#CCO_B0000
000>

SELECT ?name ?type
FROM <http://www.cellcycleontology.org/ontology/rdf/CCO>
WHERE{
  query_term_id: rdfs:label ?name.
  query_term_id: rdf:type ?type.
}
```

The separation of the two triples by a point implies the logical 'AND'. This choice requires that every term must have a name and a type, indicated with rdfs:label and rdf:type (see Table 3.3). No results will be returned for terms that are lacking either the name or the type. On the other hand, if there is more than one name or more than one type, there would be more than one result returned, leading to ambiguity at the level of the Java code for the visualization. The assumption behind this query is that every term has exactly one name and one type. This holds for CCO and its RDF translation, but it may not hold for any random RDF knowledge base.

The query might be formulated slightly more generically through the UNION and the OPTIONAL operators, which would still return some useful results in the case there are some flaws in the RDF graphs. However, this would not lead to a predictable query system. We must conclude that constraints and guidelines for creating RDF representations are necessary in order to create larger interoperable RDF systems. Such guidelines are provided by the W3C standards and recommendations. They turn out to be very valuable here for a very basic query. Also the rest of the work in this thesis will show that standards are of the greatest importance in the field of Knowledge Management.

Get the neighborhood for horizontal browsing

The knowledge base visualization tab of the DIAMONDS platform was required to show all the neighboring nodes in the ontology graph when a node was clicked. This action translates to a query for the union of all the outward relation arcs (outward arrows) and the inward relation arcs (inward arrows):

```
# NAME: get_neighborhood
# PARAMETER:
# http://www.cellcycleontology.org/ontology/rdf/CCO:
# The default ontology URI
# PARAMETER: CCO_B0002494:
# The term for which you get the neighboring terms and the
# relations to them
# FUNCTION: returns the neighborhood of a term

PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX query_term_id:
<http://www.cellcycleontology.org/ontology/rdf/CCO#CCO_B0002
494>

SELECT ?outward_arrow ?object_neighbor ?inward_arrow
?subject_neighbor
FROM <http://www.cellcycleontology.org/ontology/rdf/CCO>
WHERE {
  {
  query_term_id: ?outward_arrow ?object_neighbor.
  ?object_neighbor rdf:type ?x
  }
UNION
  {
  ?subject_neighbor ?inward_arrow query_term_id:.
  ?subject_neighbor rdf:type ?y
  }
}
```

The query also contains the limitation that the arcs should connect to other terms and not to metadata like the name, the definition and comments. This is expressed by requiring that the neighbor nodes should be of a certain type, indicated with the predicate ‘rdf:type’. This practice was used for the RDF translations used for CCO. The intended use of ‘rdf:type’ is to connect an instance with a class to which the instance belongs. However,

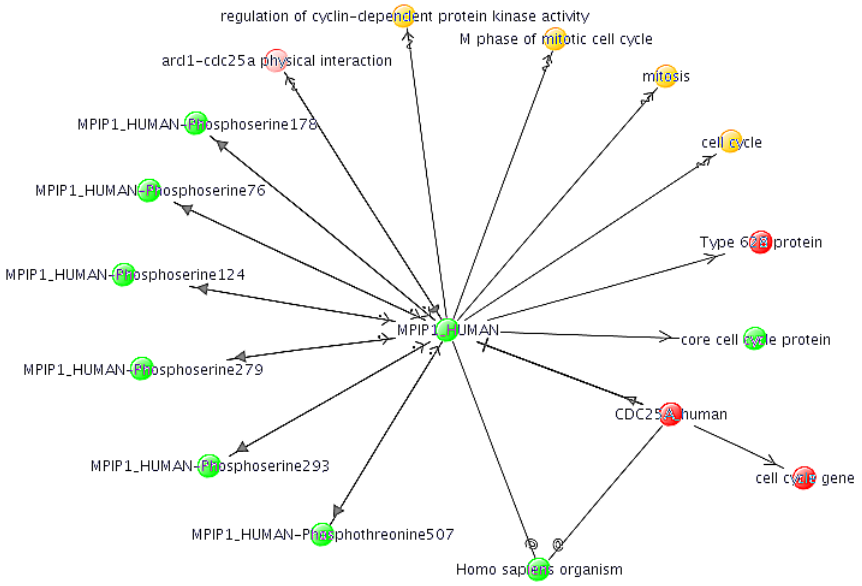


Figure 4.4: The neighborhood of the protein class CDC25A. The colors of nodes represent the type of visualized term (indicated as the object in an RDF triple 'term - rdf:type - object'). The symbols on the arcs between the nodes represent the relation type (indicated as the predicate in an RDF triple 'term1 - relation type - term2').

all the terms in CCO are classes instead of instances, which might make the usage of 'rdf:type' not in accordance with the Direct Semantics of OWL. But since RDF is not a decidable Description Logic, and therefore not be directly useful for fully automated reasoning anyway, such modelling choices remain unproblematic. For the purpose of SPARQL querying, this modelling feature proves to be useful for distinguishing the terms in CCO from other subject and object nodes in the RDF graph. The visualization of a neighborhood query can be seen in Figure 4.4.

Search the knowledge base on a name

The presentation of the root of the ontology to the user is one method to start browsing a knowledge base. Another method, that is often more useful for lay users, is to search for the terms by their name or by a part of the name.

SPARQL offers the possibility to query on regular expressions, which are formal expressions to identify a series of text characters. The following query was used as an initialization query to give the user access to CCO in the visualization tab in the DIAMONDS platform:

```
# NAME: search_labeled_terms_on_name
# PARAMETER:
# http://www.cellcycleontology.org/ontology/rdf/CCO:
# The default ontology URI
# PARAMETER: cell: the first search-string
# PARAMETER: activity: the second search-string
# FUNCTION: returns all the labeled terms for which the name
# contains 'cell' and 'activity'

PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>

SELECT ?term_id ?name ?type
FROM <http://www.cellcycleontology.org/ontology/rdf/CCO>
WHERE {
  ?term_id rdfs:label ?name.
  ?term_id rdf:type ?type.
  FILTER regex(?name, 'cell','i')
  FILTER regex(?name, 'activity','i')
}
```

The regex operator takes a literal variable as first parameter and a regular expression as second parameter. If the literal matches the regular expression, the regex operator returns the boolean value 'true' to the FILTER operator. Literals that do not return a 'true' via the regex operator do not bind the FILTER statement and will not appear in the answer set. The query shown here is case insensitive, which is achieved through the optional third parameter 'i'. This means that the user does not have to know the correct spelling exactly and whether upper case or lower case is used. This increases the chance that he will get the desired results.

Get the properties for information

Another important requirement for the knowledge base navigation tab in the DIAMONDS platform is the query for all the properties (the metadata) of a certain term. Such a query must be written very specifically on the RDF model used, which is the OBO to RDF translation of ONTO-PERL in

this case. It collects all the different properties as a union of different triple patterns, one for each property. Some properties consist of parts that are always required and other parts that are optional. This results in a rather long query. The number of results is however low for CCO, for instance 30 entries, which makes the series of unions very manageable for the SPARQL engine. The different sections in the query were also delivered separately for inclusion in the Java code. This is the query for returning the properties of a term:

```
# NAME: get_properties
# PARAMETER:
# http://www.cellcycleontology.org/ontology/rdf/CCO:
# The default ontology URI
# PARAMETER: CCO_B0002494:
# The term for which you want the properties
# FUNCTION: returns the properties of a given term

PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX default_ontology:
<http://www.cellcycleontology.org/ontology/rdf/CCO#>
PREFIX query_term_id:
<http://www.cellcycleontology.org/ontology/rdf/CCO#CCO_B0002
494>

SELECT ?name ?alt_id ?definition ?dbname_def_xref ?accession_
def_xref
?comment ?synonym ?scope ?dbname_syn_xref ?accession_syn_xref
?dbname_term_xref ?accession_term_xref
FROM <http://www.cellcycleontology.org/ontology/rdf/CCO>
WHERE{
  {query_term_id: rdfs:label ?name}
UNION{
  query_term_id: default_ontology:hasAlternativeId ?alt_id.
  }
UNION{
  query_term_id: default_ontology:Definition ?a.
    {?a default_ontology:def ?definition.}
  UNION{
    OPTIONAL{
      ?a default_ontology:DbXref ?b.
      ?b default_ontology:dbname ?dbname_def_xref.
      ?b default_ontology:acc ?accession_def_xref.
      }
    }
  }
```

```
UNION
  {query_term_id: rdfs:comment ?comment}
UNION{
  query_term_id: default_ontology:synonym ?a.
  ?a default_ontology:syn ?synonym.
  OPTIONAL{?a default_ontology:scope ?scope.}
  OPTIONAL{
    ?a default_ontology:DbXref ?b.
    ?b default_ontology:dbname ?dbname_syn_xref.
    ?b default_ontology:acc ?accession_syn_xref.
    }
  }
UNION{
  query_term_id: default_ontology:xref ?a.
  ?a default_ontology:dbname ?dbname_term_xref.
  ?a default_ontology:acc ?accession_term_xref.
  }
}
```

Show the path to the root

The DIAMONDS platform allows to right-click on the terms and show a limited list of possible queries. One of these possibilities is to show all the intermediate terms between the clicked term and the root of the ontology. This is the query that essentially answers the question: "What is this term?"

This is a very challenging query because the path to the root does not have a fixed length. Moreover, it is possible that multiple paths to the root exist and all these paths may be useful to show to the users. Handling such a variability is an easy task for procedural languages like Java or Perl, but not for SPARQL. An easier query that is related to the query for the path to the root is to ask for all the ancestors of a certain term. The difference with the query for the path to the root is that any *is_a* relations are missing in the set that only contains all the ancestors. Also an RDF graph that contains all the logical inferences (containing an *is_a* relation arc for every ancestor - descendant pair) is not useful for this query, because we want to show only the paths that were designed by the ontology engineers. Too many logically inferred relation arcs result in a chaotic visual representation.

The only solution to this problem is to assume a very long path to the root with fixed numerical variable names (n1, n2, etc.) for all the intermediate terms in the path. However, the query cannot require that such a long path exists, because shorter paths must also return results. Any trailing part of the path must be made optional for that reason. This has resulted in a very long SPARQL query, with many OPTIONAL clauses:

```
# NAME: get_labeled_hierarchy_to_root
# PARAMETER: The default ontology URI:
# http://www.cellcycleontology.org/ontology/rdf/CCO:
# PARAMETER: CCO_T0000004:
# The term for which you will get the hierarchy to the root
# FUNCTION: returns all the possible labeled subsumption
# paths to the root

PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX default_ontology:
<http://www.cellcycleontology.org/ontology/rdf/CCO#>
PREFIX query_term_id:
<http://www.cellcycleontology.org/ontology/rdf/CCO#CCO_T0000
004>

SELECT *
FROM <http://www.cellcycleontology.org/ontology/rdf/CCO>
WHERE {
  OPTIONAL{
    query_term_id: default_ontology:is_a ?n1 .
    ?n1 rdfs:label ?n1_name.
    ?n1 rdf:type ?n1_subnamespace.
  }
  OPTIONAL{
    ?n1 default_ontology:is_a ?n2 .
    ?n2 rdfs:label ?n2_name.
    ?n2 rdf:type ?n2_subnamespace.
  }
  OPTIONAL{
    ?n2 default_ontology:is_a ?n3 .
    ?n3 rdfs:label ?n3_name.
    ?n3 rdf:type ?n3_subnamespace.
  }
  OPTIONAL{
    ?n3 default_ontology:is_a ?n4 .
.
.
.
    ?n36 rdf:type ?n36_subnamespace.
  }
  OPTIONAL{
    ?n36 default_ontology:is_a ?n37 .
    ?n37 rdfs:label ?n37_name.
    ?n37 rdf:type ?n37_subnamespace.
  }
}
```

A number of 37 is the maximum that is accepted by the SPARQL engine of Virtuoso. This is a reasonable depth for querying ontologies because they are mostly less than 10 steps deep. CCO has an average depth of around 8 and a maximum depth of 34 for the long taxonomic branch of the term '*Homo sapiens* organism'.

Also an inferred model with inferences for transitivity cannot make this query shorter, nor the support for transitivity in query time. Both of these solutions would return a mixture of superclasses in parallel paths in a random order. After all, SPARQL uses a two-dimensional format with rows and columns for representing the result set. In order to produce the same results with a shorter query and for arbitrarily many paths of arbitrary lengths, SPARQL would require an extension that makes also the number of columns dependent on the outcome of the query, which is done now only for the number of rows. SPARQL 1.1 has extended SPARQL 1.0, which was used here, with abilities for reasoning with transitivity in query time for a single path, but not to a flexible number of columns for recursive queries over arbitrarily many paths of unknown lengths. The current situation could be improved by support for path lengths higher than 37.

Many SPARQL queries that contain more than 10 lines become very slow and often the SPARQL engine even warns immediately that the query is too heavy to compute. After all, in the worst cases SPARQL requires a polynomial amount of memory space and an exponential amount of time in function of the size of the query. However, this query with 37 clauses and more than 100 lines always responds within some seconds, even for very long paths. For our human reasoning intelligence, this is not a surprise, since this query is not a worst case at all. If only the engine starts tackling the problem from the bottom of the path, it will never need to store anything in the memory that is not of use for the further calculation. Clearly, the SPARQL engine of Virtuoso handles this query in such a way.

4.3.4 SPARQL queries of biological interest

The standard queries for visualization in the previous section have the potential to be useful in any scientific area that uses ontologies in RDF for knowledge representation. However, the final goal of the DIAMONDS project was the creation of a computational platform that was specifically oriented towards cell cycle research. Biologist researchers have research questions that do not translate to such standard SPARQL queries.

Find a chain of interactions between two protein types

An important research question for genome biologists is to find interaction pathways. A biological pathway consists of a series of proteins and protein complexes that interact with each other in a certain order. This results in a certain cellular process with a certain function, which can be upregulated or downregulated by the expression of the genes that encode the proteins participating in the pathway. The regulation of the whole cell cycle process happens through a large number of such pathways. By finding and describing these pathways, cell cycle researchers get a better understanding of the basic machinery that is shared by most living species on earth.

The basic components of biological pathways are the interactions between the proteins in the pathway. For this reason the research question often reduces to the more basic question of finding these protein-protein interactions (PPI) and chains of protein-protein interactions. In CCO, every interaction between any protein type A and any other protein type B is modeled as a separate term, which represents the class of all such A-B interactions (cfr. any of the terms at the top of Figure 4.1). The following SPARQL query retrieves all the chains of interactions between a protein type 1 and a protein type 4, going over two intermediate protein types 2 and 3:

```
# NAME: get_interaction_paths
# PARAMETER:
# http://www.cellcycleontology.org/ontology/rdf/Sc:
# The default ontology URI
# PARAMETER: CCO_B0003246: protein 1, one of the proteins
# for which you look for a path of interactions
# PARAMETER: CCO_B0000427: protein 4, the other protein for
# which you look for a path of interactions
# FUNCTION: returns all the paths with the structure:
# given protein -> interaction1 -> protein2 -> interaction2
# -> protein3 -> interaction3 -> other given protein

PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX default_ontology:
<http://www.cellcycleontology.org/ontology/rdf/Sc#>
PREFIX protein1: <http://www.cellcycleontology.org/ontology/
rdf/Sc#CCO_B0003246>
PREFIX protein4: <http://www.cellcycleontology.org/ontology/
rdf/Sc#CCO_B0000427>

SELECT *
```

```
FROM <http://www.cellcycleontology.org/ontology/rdf/Sc>
WHERE {
  # The path:
  protein1: default_ontology:participates_in ?interaction1.
  ?protein2 default_ontology:participates_in ?interaction1.
  ?protein2 default_ontology:participates_in ?interaction2.
  ?protein3 default_ontology:participates_in ?interaction2.
  ?protein3 default_ontology:participates_in ?interaction3.
  protein4: default_ontology:participates_in ?interaction3.

  # Filtering out false and duplicate paths:
  ?interaction1 rdf:type default_ontology:interaction.
  ?interaction2 rdf:type default_ontology:interaction.
  ?interaction3 rdf:type default_ontology:interaction.
  filter(?interaction1!=?interaction2)
  filter(?interaction1!=?interaction3)
  filter(?interaction2!=?interaction3)
  filter(protein1:!=?protein2)
  filter(protein1:!=?protein3)
  filter(?protein2!=?protein3)
  filter(?protein2!=protein4:)
  filter(?protein3!=protein4:)
}
```

The query needs to filter out many false and duplicate paths. This is because SPARQL does not require that different parameters in the triple patterns also refer to different identified resources. Indeed, the query with a triple pattern consisting of a single triple (?s ?p ?o) will return all the possible triples in the store, also the triples where the subject and the object are identical. On the other hand, the query consisting of the pattern (?s ?p ?s) will return just those triples where the subject and the object are identical. Filter operations are required to retrieve just those triples where the subject and the object are different. This applies equally to triple patterns that consist of different lines, like for the SPARQL query above. Obviously, we are not interested in paths where the same protein type occurs more than once. In order to express this, many filter lines are required.

The problem with this query was the bad performance of the computation. Four proteins and three intermediate interactions was the maximum path size that the SPARQL engine could handle. The query consumed a lot of time, up to one hour, and longer paths caused the immediate rejection of the query through an error report that announced an expected query time that was too long. The bad performance of this query is due to the exponential relation between the length of the path and the number of possible

paths that have to be investigated. Each extra node in the path multiplies the number of possible paths with a factor that depends on the sparseness of the PPI network.

Experiments succeeded only on the smaller CCO files for a single model species, but not on the integrated CCO. Finally, retrieving the labels of the terms in the same query increased the number of lines, which made the performance metrics even worse. These hurdles have made it practically very difficult to bring the query to a level where expert biologists could assess the biological relevance of the results. Because of the bad performance, the query could not be included as a standard lookup option for proteins in the DIAMONDS platform.

An interesting possibility to address this research challenge might consist of the creation of recursively precomputed relations between any two proteins that reveal more information about the nature of the interaction chains between them. This could overcome the computational hurdles for answering queries about protein interactions. However, such investigations would require substantial insight in PPI networks and more biological expertise to assess the relevance of the outcome.

Find the orthologs of a protein type

There are – of course – many other research questions that CCO wants to support. Most of these question do not depend so heavily on the computational performance of a SPARQL query engine as compared to the question of finding chains of protein interactions. The creation of information about orthology is an example of a computational effort that needs to be executed before the launch of any query. It is computed with BLAST (Basic Local Alignment Search Tool) [154], which compares genome sequences from the different species in detail – at the level of the individual base pairs in the sequences – and by finding regions of local similarities between the sequences. An all-against-all BLAST matrix of compared genes is the input for the clustering algorithm OrthoMCL, which generates clusters of putative orthologs.

The proteins that were supposedly orthologous with any core cell cycle proteins were integrated in CCO by linking them all to newly created terms for the orthology clusters. The clusters are generated from a large amount of input data and they are dependent on the particular settings (default settings in this case) of the OrthoMCL algorithm. The only reasonable option was to use a fully automated naming system for them, using num-

bers. This practice is not problematic for CCO, because it is considered as an application ontology, instead of a domain ontology that is supposed to use only terms that are universally accepted in a certain scientific domain. The orthology terms are classes of proteins and they are superclasses of all the typical protein types that are known by biologists through a meaningful name.

The orthology cluster terms themselves are not supposed to be returned as an answer to a query, because their names will be meaningless for biologist users. A meaningful query will retrieve the subclasses of these terms, which are all classes of orthologous proteins. These subclasses are the protein types with a meaningful name that is known by biologist experts. They group the proteins that belong to a specific organism. The following SPARQL query retrieves the orthologs of a given protein (the query term) together with the organism to which it belongs:

```
# PARAMETER:
# http://www.cellcycleontology.org/ontology/rdf/CCO:
# The default ontology URI
# PARAMETER: CCO_B0002268:
# The term for which you want the orthologs
# FUNCTION: returns all the orthologs for a given protein
# and The organism to which they belong

PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX default_ontology:
<http://www.cellcycleontology.org/ontology/rdf/CCO#>
PREFIX query_term_id: <http://www.cellcycleontology.org/
ontology/rdf/CCO#CCO_B0002268>
SELECT ?ortholog_id ?ortholog_name ?organism_id
?organism_name
FROM <http://www.cellcycleontology.org/ontology/rdf/CCO>
WHERE{
  query_term_id: default_ontology:is_a ?ortholog_cluster_
  protein.
  ?ortholog_id   default_ontology:is_a ?ortholog_cluster_
  protein.
  ?ortholog_cluster_protein   rdf:type default_ontology:term.
  ?ortholog_id   rdf:type              default_ontology:
  protein.
  ?ortholog_id   default_ontology:belongs_to ?organism_id.
  ?ortholog_id   rdfs:label            ?ortholog_name.
  ?organism_id   rdfs:label            ?organism_name.
  FILTER(?ortholog_id != query_term_id:).
}
```

This query is very suited as a standard query in visualization systems for genome biologists who are comparing genomes of different species.

User-defined queries

It is impossible to present all the expressivity of the SPARQL query language through mouse-clickable drop-down menus and tool bars. That is why the DIAMONDS platform was also given a special tab, separate from the visualization tab, for launching user-defined SPARQL queries. This allows, in theory, to query about every biologically relevant aspect of the knowledge that is integrated in CCO. However, in practice this task cannot be expected from biologists. The development of the library of queries and BioGateway later on was inspired by the research problem of bridging the gap between SPARQL and the users who want answers on biological questions. The maintenance of the DIAMONDS platform was finally abandoned in order to concentrate on the development of BioGateway.

4.4 Benchmarking biomedical SPARQL queries

4.4.1 Introduction

All the queries that were written for the DIAMONDS platform and CCO were developed within Virtuoso, one of the major RDF stores that have an open source version. Virtuoso has both a commercial version and an open source version and these products have been improved and extended with regular releases during the last years. It has become faster, more scalable for larger amounts of data and more up-to-date with the latest W3C recommendations, like SPARQL/Update, federated querying and support for reasoning. Other server-based software solutions for RDF have been going through a similar evolution. For this reason, it is important to have an objective benchmark for comparing the performance of RDF software.

There are several benchmarks for RDF that are testing the latest releases of RDF software at regular intervals:

- The Berlin SPARQL Benchmark (BSBM) [155] has created artificial RDF data about commercial products as well as a library of SPARQL queries that customers and salesmen might need for finding the right information. This library reviews all the aspects of RDF and SPARQL, like UNIONs, OPTIONALs and FILTER op-

erations. BSBM compares native RDF stores, Named RDF graphs, relational databases that expose RDF and SPARQL-wrappers around other data-structures.

- The Lehigh University Benchmark (LUBM) [156] assesses several performance metrics through a set of 14 SPARQL queries on an artificial domain ontology about universities, departments and students, that is used to annotate customizable, artificial data. The idea is to evaluate Semantic Web repositories, like RDF and OWL stores.

- The DBpedia SPARQL Benchmark [157] compares the performance of RDF stores with real data from DBpedia [158], which is a project that aims to turn data from Wikipedia into RDF.

Artificially created data and benchmarks are very well suited to test and compare every aspect of W3C's specification for RDF and SPARQL. However, there is one obvious disadvantage: in testing all the challenging features of the language, the benchmark may not be oriented enough on the real needs of end-users. Moreover, the query needs of biomedical scientists may not be comparable to the query needs of salesmen and customers. For these reasons, it would be useful to create a benchmark with real, biomedical RDF data and queries.

4.4.2 The NTNU benchmark

The RDF format of CCO and the queries that have been developed for CCO and the DIAMONDS platform were used to create the NTNU benchmark [4, 159]. The historical development of the NTNU benchmark is represented in Figure 4.5. This development started with the engineering of CCO in the OBO format. The contents of CCO were selected on the basis of feedback that was provided by biology experts who had questions about the cell cycle. The RDF format was created later as an export from the OBO format. This RDF format was loaded in a Virtuoso RDF store, which has enabled the development of a library of SPARQL queries for CCO. The export specification for converting the OBO format to an RDF format has been improved later by feedback stemming from the experiences of querying CCO in Virtuoso.

Apart from providing a basis for the comparison of the performance of different RDF software, the library of queries that was engineered and

Graph	Number of triples	Number of classes	Depth	Average depth	Number of relations	Number of relation types
cco	2503040	89526	33	7.72	461946	30
cco_tc	3170556	89526	33	7.72	1129462	30
cco_A_thaliana	356903	12578	34	9.11	22132	30
cco_A_thaliana_tc	469484	12578	34	9.11	134713	30
cco_S_cerevisae	842344	35004	34	7.99	171825	30
cco_S_cerevisae_tc	1120545	35004	34	7.99	450026	30
cco_S_pombe	406131	14584	34	8.86	39997	30
cco_S_pombe_tc	533481	14584	34	8.86	167347	30
cco_H_sapiens	836622	29187	34	8.29	121383	30
cco_H_sapience_tc	1076760	29187	34	8.29	361521	30

Table 4.1: The statistics of the 10 RDF graphs that are used in the NTNU benchmark. The graph names ending on '_tc' contain triples that were logically inferred from closure rules, like the transitivity of subsumption. The depth of a graph refers to the maximum number of superclasses for a class in the hierarchy, whereas the average depth is the number of superclasses averaged over all the classes.

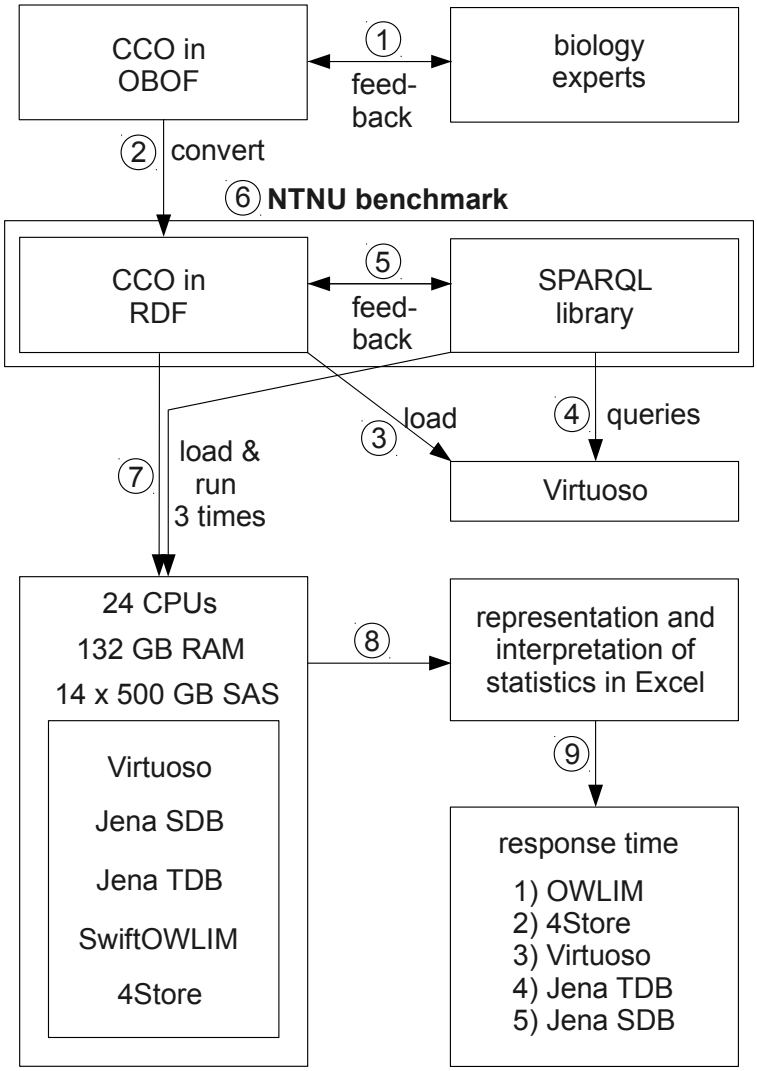


Figure 4.5: The historical development of the NTNU benchmark, represented in nine steps. The interpretation of the response times have to be done carefully, since the material for the benchmark, which consists of RDF data and a library of SPARQL queries, was developed and optimized on a Virtuoso RDF store.

optimized for Virtuoso can also shed light on how well the different software implementations have used the same standards for RDF and SPARQL. For this reason the ten RDF graphs with content from CCO and the library of SPARQL queries were selected for the NTNU benchmark.

Four graphs contain taxon-specific knowledge about the cell cycle, corresponding to the four species taxa that were used for CCO. A fifth graph is the RDF translation of the whole CCO, which integrates the other four and contains additional orthology data that cross-links the other graphs. Each of these five graphs has a duplicate that contains inferred knowledge from logical rules like transitivity of relations, in addition to the original data. The ten RDF graphs and statistics about them are given in Table 4.1.

All 24 queries of the SPARQL library were selected without any modifications. These queries were characterized on the basis of nine features:

- *Filter* is used to filter a result set on numerical constraints.
- Having more than 8 triples in the query pattern (> *8 query triples*) was considered as a special feature.
- *Optional* may return extra variables in an optional query pattern, without restricting the result set in case the query pattern cannot be filled in.
- *Limit* is used to limit the result set to a given number of results.
- *Order by* is an operator that orders the result set alphabetically on a given variable.
- The *distinct* operator reproduces every result only once within a result set.
- *Regex* provides a method to filter on regular expressions.
- The *union* operator combines the result sets of different query patterns.
- *Count* can be used to count the number of results in the result set.

This characterization is shown in Table 4.2. It should be remarked again that the queries were never designed to include special combinations of the nine features, which is a common practice in the design of query

	Filter	*> 8 query triples*	*Optional*	*Limit*	*Order by*	*Distinct*	*Regex*	*Union*	*Count*
Q1						x			
Q2								x	
Q3	x	x	x			x	x		
Q4									
Q5									
Q6									
Q7	x						x		
Q8						x			
Q9	x								
Q10	x	x				x	x		
Q11									
Q12								x	
Q13		x	x			x		x	
Q14					x				
Q15									
Q16									
Q17						x			x
Q18	x	x			x		x	x	
Q19						x		x	x
Q20						x			x
Q21						x			x
Q22									
Q23									
Q24	x			x			x		

Table 4.2: 24 queries (Q1 through Q24) were used in the benchmark. Special SPARQL features for every query are listed in this table.

benchmarks. The advantage of this choice is that the NTNU benchmark reflects the query needs of biologists more realistically.

4.4.3 Comparing RDF software

The selection of the contents of the NTNU benchmark is followed by the execution of a comparison that uses the benchmark. Four other non-commercial RDF storage software solutions were compared to the performance of the open source edition of OpenLink Virtuoso:

- Jena SDB
- Jena TDB
- SwiftOWLIM
- 4Store

Jena [160] is an open source software layer that has been implementing the Semantic Web recommendations since the earliest developments of RDF and OWL. The names 'SDB' and 'TDB' can be read as SPARQL Data Base and Triple Data Base, however, the acronyms are chosen as alphabetic successors of 'RDB', the acronym for the classical Relational Data Base. 4Store [161] and OWLIM [162] have become popular much later, around 2010. At the time of the comparison presented here, which was executed in 2010, OWLIM was released both as an open source version, called SwiftOWLIM, and a closed source commercial version, called BigOWLIM. In order to assess the solutions that are available to science, the open source version of OWLIM was selected for the comparison.

A special procedure was developed for executing the comparison with the NTNU benchmark. The experience of querying through SPARQL on Virtuoso has revealed that the response time of a query may be heavily dependent on the execution of previous queries and the size of the RDF data that is loaded in the store. Virtuoso contains an optimizer for SPARQL that decides which search strategy should be performed first. In addition to that, it also caches results, partial results and RDF triples that are of importance for producing a quick answer to queries that were launched before. In order not to be confronted with the unpredictable behavior that stems from such optimizations, the comparison procedure included three phases

for each RDF software solution. Data files that were produced during a certain phase were deleted and the RDF store was shut down before the start of the next phase. This implies that also the files with the RDF data about CCO were deleted each time.

During each phase, all the queries in the SPARQL library were launched once for each of the ten RDF graphs and always in the same order. The outline of the testing procedure is formulated more formally as follows:

```
for each RDF storage system {
        repeat 3 times {
                delete data files
                switch on RDF store
                load data
                run all queries
                switch off RDF store
        }
}
```

The analysis was performed on a Dell R900 machine with 24 Intel(R) Xeon(R) CPUs (2.66GHz), operated by Unix. The machine was equipped with 132GB main memory and 14x500GB 15 000 RPM SAS hard drives. This machine was, however, not a machine that was specifically dedicated for this task. Other processes, from other computational tools, were also running on the same server. The three replicated experiments have enabled to compare differences in the run-to-run reproducibility. For any strategies to improve the performance of the query execution, like indexing, the defaults suggested by each of the RDF storage systems have always been selected.

4.4.4 Analysis and results

The loading performance and the total query time required by the 5 storage systems can be found in Table 4.3. The times for loading the 11 million triples varies with a factor 15 for the different systems: 4Store could load the 10 graphs in 2 minutes and 8 seconds, whereas Jena SDB took more than 33 minutes for this task. Considering the loading time as a criterion for the performance, we get the following order: 4Store - OWLIM - Virtuoso - Jena TDB - Jena SDB.

The total time required for answering all the queries in the NTNU

benchmark suggests an entirely different assessment. Whereas Virtuoso uses only 2 minutes and 24 seconds to answer a set of 240 queries, OWLIM and 4Store need about 4 hours and 13 hours respectively to answer the same set. Jena SDB on the other hand, being the slowest system for loading, needs only 12 minutes and 10 seconds for this task. However, the ensuing analysis will show that the total loading time is not a correct criterion to assess the performance of the storage systems.

Store	Load time	Total Query time
4Store	128 s	47567 s
OWLIM	230 s	14258 s
Virtuoso	527 s	204 s
Jena TDB	833 s	1446 s
Jena SDB	2005 s	730 s
AVG	745 s	12841 s

Table 4.3: Load time - the total time for loading the 10 graphs (11.3 million triples) averaged over the three runs. Total Query time - The total time for answering the 240 queries (24 queries on 10 graphs) averaged over the three runs.

The 24 queries of the benchmark were run 3 times on the ten graphs for 5 RDF storage systems. All the queries were launched from the command line of the Unix operating system, which gave detailed reports about the execution time of the queries by comparing precise Unix dates. Each query was run on the 10 RDF graphs, however, the differences in execution time for the same query on different RDF graphs have not appeared to be very interesting nor unpredictable. Since these differences are not of importance for comparing the 5 storage systems, they were summed to a single time value. This summation includes any delay time between the execution of different queries. The run-to-run reproducibility of the execution time during the 3 phases was very good. Obviously the deletion of data files and the restart of the system between the phases has guaranteed this reproducibility. The values were averaged, reducing the result sheet to 120 values that are the basis of the analysis.

These 120 values can be found in Table 4.4, with the queries Q1 to Q24 as rows and the type of storage system as columns. They are a summation of time values for a query that has run on ten graphs, expressed in

Query	Virtuoso	OWLIM	4Store	Jena TDB	Jena SDB	Geom. avg.
Q23	5.630	0.009	1.343	10.454	13.343	1.568
Q16	5.617	0.009	1.346	10.825	13.345	1.579
Q11	5.343	0.011	1.419	10.703	13.339	1.641
Q15	6.163	0.018	1.390	10.544	13.342	1.850
Q4	5.916	0.017	1.539	10.773	13.348	1.859
Q12	7.198	0.030	1.400	10.731	13.373	2.125
Q13	1.658	0.034	1.569	14.156	52.545	2.310
Q8	7.094	0.049	1.577	10.449	13.336	2.380
Q2	7.281	0.052	1.438	10.768	13.337	2.391
Q22	5.170	0.173	1.428	10.981	13.709	2.862
Q5	2.639	0.408	1.526	11.000	13.446	3.000
Q19	2.065		1.326	9.795	13.390	4.353
Q9	5.820	1.067	2.133	10.699	13.711	4.546
Q6	4.054	2.020	1.779	10.573	14.523	4.676
Q21	3.379		1.316	9.818	13.335	4.912
Q17	5.648		1.350	10.119	13.390	5.669
Q1	1.897	8.024	1.647	14.258	18.064	5.781
Q20	6.110		1.315	10.686	13.387	5.822
Q10	4.679	4.664	2.529	11.676	13.757	6.159
Q14	5.775	91.894	1.401	46.433	13.338	13.57
Q24	2.813	27.242	14.366	38.619	24.719	16.007
Q7	2.617	28.996	14.110	39.248	26.519	16.196
Q3	3.358	1121.702	3.654	27.049	30.476	25.761
Q18	22.840	8325.734	24999.569	493.596	76.013	708.373

Table 4.4: A cumulative response time in seconds for each query, summed over the 10 graphs, compared for the five RDF storage solutions. The queries are ordered from fast to slow, based on the geometrical average performance (Geom. avg.). The syntax of some queries could not be processed by OWLIM.

seconds. This means that, for example, an execution of query Q5 by Jena TDB on a single of the 10 RDF graphs took about 1.1 seconds (11 seconds for all the graphs). The geometrical average time value for each query, averaged over the 5 types of storage systems, is represented in the last column. The benefit of representing the geometrical average instead of the normal average is that values that are very small compared to others are also taken into account. This average value has been used for ordering the queries from fast to slow. Query Q23 has appeared to be the fastest responding query.

OWLIM was the only storage system that could not interpret the syntax of some SPARQL queries in the library, namely Q17, Q19, Q20 and Q21, which are queries that include the COUNT operator. All the other systems could interpret all the queries that were tested and developed in Virtuoso. Also the loading of the 10 RDF files into the different systems had not posed problems. This means that the W3C standards for SPARQL and RDF were implemented quite well.

The exceptionally long response times for queries Q3 and Q18 suggest that OWLIM and 4Store were not optimized to handle these queries efficiently. The 25 000 seconds that 4Store needed for query Q18 explains its long total query time. Four of the five slowest queries contain a FILTER REGEX operator and two of the five use the ORDER BY operator. Query Q18, the most problematic query, contains two FILTER REGEX operators, an ORDER BY and uses more than 8 triples in its query pattern, as can be seen in Table 4.2. The whole query Q18 reads as follows:

```
# NAME: search terms by properties
# PARAMETER: cell: first search-string
# PARAMETER: cycle: second search-string
# FUNCTION : returns terms with the name, definition,
#            synonym or comment containing the specified
#            search strings

BASE   <http://www.semantic-systems-biology.org/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ssb: <SSB#>
PREFIX graph: <CCO>

SELECT ?name ?found_in ?type_of_found_text
WHERE {
 GRAPH graph: {
 FILTER regex(str(?found_in), 'cell', 'i')
 FILTER regex(str(?found_in), 'cycle', 'i')
```

```
  ?term_id rdfs:label ?name .
  {
   ?term_id ?type_of_found_text ?found_in .
   ?term_id rdfs:label ?found_in .
  }
  UNION {
   ?term_id ?type_of_found_text ?a .
   ?term_id ssb:Definition ?a .
   ?a ssb:def ?found_in .
  }
  UNION {
   ?term_id ?type_of_found_text ?a .
   ?term_id ssb:synonym ?a .
   ?a ssb:syn ?found_in .
  }
  UNION {
   ?term_id ?type_of_found_text ?found_in .
   ?term_id rdfs:comment ?found_in .
  }
 }
}
ORDER BY ?term_id
```

A graphical representation of the results is shown in Figure 4.6. The comparison of the performance of the 5 storage systems has to be analyzed with care, because the queries were optimized to perform well on Virtuoso and were never tested before on the other systems. However, the execution time of the 11 fastest queries is very consistent. The five systems always have the same ranking for these 11 queries. For fast queries, OWLIM responds 30 to 100 times faster than 4Store, 100 to 500 times faster than Virtuoso and up to 1000 times faster than Jena SDB and Jena TDB. The graph shows that OWLIM is the only system that has continued to optimize the speed of the queries into the range of milliseconds. 4Store got second place, whereas Virtuoso, the store on which the queries were optimized, comes third. In the fourth and fifth places come Jena TDB and Jena SDB.

Slow queries are heavily biased in favor of Virtuoso, since the optimization process has eliminated all the queries that responded extremely slow. This is why the profile of Virtuoso, shown in the middle on the graph, does not show any high peaks for the slow queries on the left side. All the queries respond within the range of seconds. It may be expected that such very long response times would be absent for the other RDF storage systems as well, if the library of SPARQL queries were optimized for these systems.

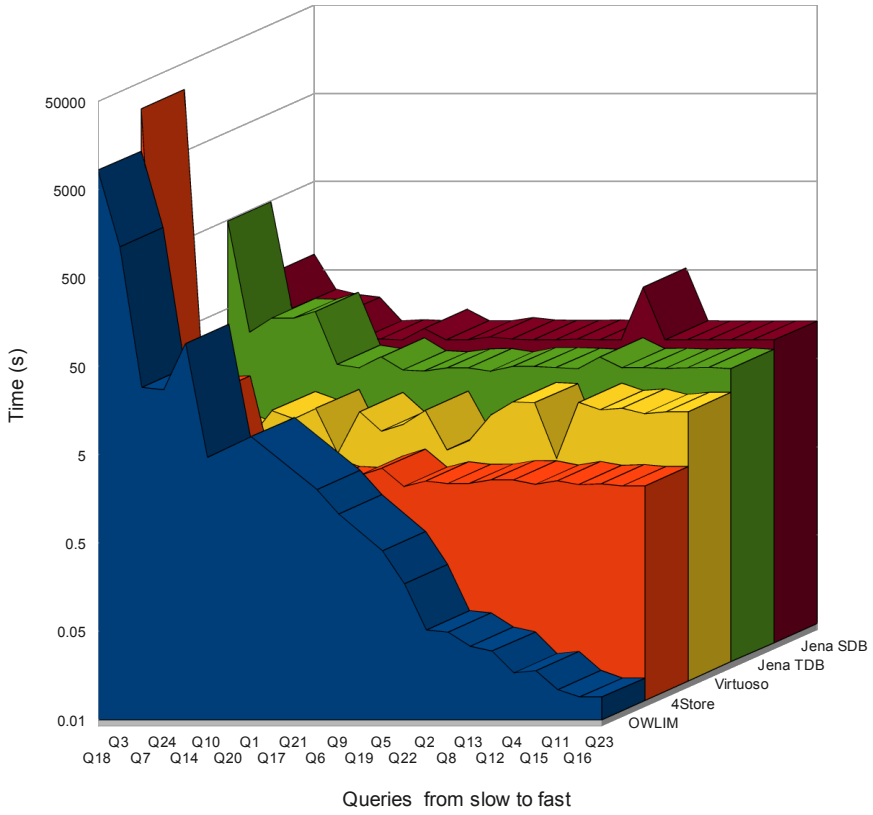


Figure 4.6: The time in seconds of queries that were launched ten times subsequently on ten graphs, compared on a logarithmic scale for five different RDF storage systems. Queries are ordered from slow to fast. OWLIM is highly optimized for fast queries, whereas the apparent good performance of Virtuoso for slow queries reflects the history of the development of the NTNU benchmark.

Chapter 5

Metarel: an ontology for relations in RDF

The knowledge in bio-medicine and many other scientific domains consists of a large number of detailed knowledge statements. These are millions of statements about classes of proteins, genes, organisms, processes, chemicals, diseases, *etc.* This brings us huge Knowledge Management challenges. RDF is very suited to represent direct relations between large numbers of classes. The use of SQL and RDF query languages on voluminous RDF data in RDBMS provides very good results for knowledge management and continues to be developed with new extensions [163, 164]. Federated querying over different RDF stores in remote locations on the Web is a promising technology that is currently explored.

Another important use case for RDF is the integration of triples in regular websites, in order to specify what the website, or parts of the website, are about. These triples refer to entities in the real world, which are made available for automated reasoners.

The incorporation of small parts of RDF code, also called RDF snippets, in websites, has been proposed for search engines and the idea is promoted by Google [165, 166]. The uncomplicated syntax and the simple idea that every triple represents a knowledge statement that is more or less context independent may lower the threshold for such applications.

In spite of the popularity of RDF, its development in the field of semantic reasoning has slowed down in favor of OWL. The standard RDF/XML syntax for OWL has enabled to load OWL ontologies in RDF stores with great ease. However, RDF has its own benefits for knowledge man-

agement and reasoning that is not exploited by OWL and its RDF syntax. Since OWL is based on DL, it describes relations between instances. The description of direct relations between classes, which form the backbone of biomedical knowledge, is not within the scope of OWL. Apart from some exceptions like subsumption and instantiation, relations that involve classes are remodeled as indirect chains of relations that are cumbersome to work with in native RDF tools. The meaning of a single RDF triple becomes dependent on other triples through the indirect model. Moreover, the OWL/RDF syntax model is not intuitive and requires the understanding of DL.

It is clear, however, that OWL is a valuable technology for reasoning and KM. Its DL-based semantics is much more expressive than what can be expressed with direct RDF relations. In this chapter we will explore how direct relations in RDF can be described semantically, without losing compatibility with logic-based OWL ontologies.

5.1 The foundation of the Metarel vocabulary

5.1.1 RDF graphs versus multidigraphs

A very short statement consisting of a subject, a predicate and and object, is a very useful element in Knowledge Representation. This is what is called a triple in RDF. It is the basis for Metarel, a controlled vocabulary about relations. The name 'Metarel' is derived from meta-relation, which is a relation between relations.

Every triple could be thought of as a labeled arc that goes from the subject to the object. The label of the arc is the predicate (typically a verb with some adverbs). Here follow some examples of triples in the biomedical domain:

- nucleus *is part of* cell
- p53 *is a* protein
- p53 *is located in* nucleus
- protein *is encoded by* gene
- heart *is part of* body
- nerve *is connected to* muscle

- horse *is a* mammal
- doctor Lisa *is employed by* UZ hospital
- doctor Lisa *takes care of* patient Marc
- UZ hospital *is located in* Ghent
- Ghent *is located in* Belgium

To clarify the approach used, we have to distinguish RDF graphs, which are represented by a set of triples, from multidigraphs (MDG), which are mathematical, directed graphs that can have multiple arcs. Every MDG can be considered as an RDF graph, but not every RDF graph is an MDG.

In an RDF graph that is an MDG (or an RDF/MDG graph), a triple consisting of three identifiers $(a\ b\ c)$ will be represented as an arc from one node to another, where the identifiers a and c will be used for the nodes, and the identifier b for the arc. Whenever an RDF graph is an MDG, we can call triples simply *relation arcs*. A graphical representation of RDF graphs and MDGs can be seen in Figure 5.1.

RDF graphs in general allow that identifiers used in the central place of a triple are also used in the first or last place of a triple (in the same triple as well as in other triples). This is not allowed in MDGs. One cannot visualize (depict) any set of triples with nodes and arcs like in an MDG. How would we visualize the triple $(a\ a\ a)$? The same problem exists for the triple $(a\ b\ c)$, when a occurs in the central place of many other triples. Moreover, RDF graphs distinguish blank nodes, which are only identified within the RDF graph, from URI nodes, which are identified within the whole Semantic Web. RDF graphs in general are harder for browsing, visualizing and querying, but they have extra possibilities that can be used to enhance the expressivity of representations in RDF.

5.1.2 The idea behind Metarel

A relation within Metarel is understood as the entire, directed connection that holds from one term to another term. Metarel aims to describe the logical meaning of relations in MDGs by using the full expressive power of RDF graphs. This is essentially different from the approach of the OWL/RDF syntax, which requires the full expressive power of RDF graphs, both for the description of relations as for the relations themselves.

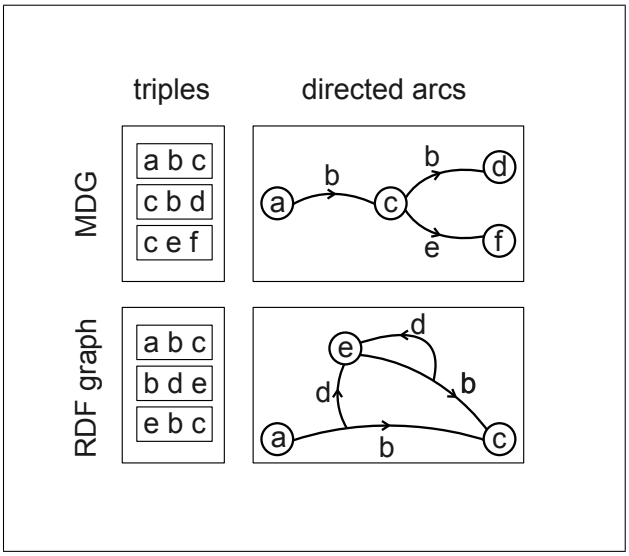


Figure 5.1: Triples and directed arcs as representations for MDGs and RDF graphs. The representation with arcs for general RDF graphs, as shown on the lower right-hand side, is not entirely satisfactory. The number of arcs does not equal the number of triples and the visualization of two triples $(a\ b\ c)$ and $(b\ a\ c)$ will either treat the triples asymmetrically or either create an infinite number of arcs.

Figure 5.2 shows the typical OWL/RDF syntax model as compared to the simple model that Metarel assumes. The OWL/RDF syntax contains a rather low fraction of meaningful classes and instances and a bigger fraction of metarelations and blank nodes. These metarelations can be logical relations (like 'subclass of' or 'instance of'), logical quantifiers (like 'some values from' or 'all values from') and range indicators (like 'on property' or 'has value'). Metarel on the other hand describes MDGs that consist entirely of meaningful classes and instances for the nodes and verbal expressions (or predicates or relation types) for the relation arcs between the nodes. Such MDGs are very intuitive in use for people who build SPARQL queries or for biologist annotators. They can use, for example, the triple 'tongue has function tasting' instead of a graphical derivative of 'tongue is a subclass of the class of all these things that are bearers of a tasting, which is a function'.

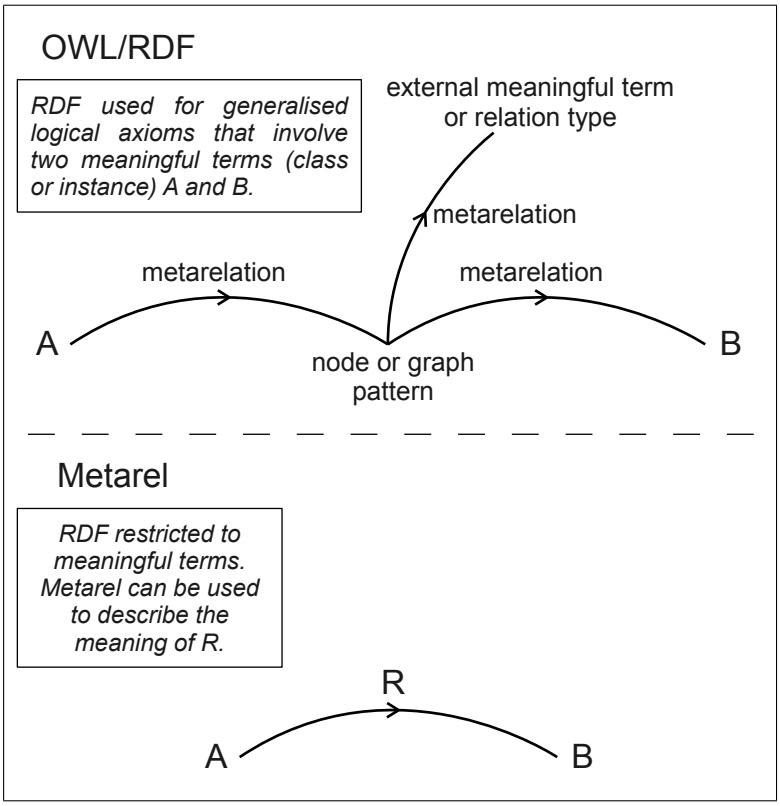


Figure 5.2: The idea behind Metarel. The OWL/RDF syntax describes advanced logical patterns whereas Metarel assumes arcs with subjects (A), relation types (R) and objects (B).

5.1.3 An MDG as a visualizable ontology representation

In order to set up a vocabulary for describing logical relations in an MDG, some clarification of terminology is required. Consider an MDG O, which allows multiple self-loops and in which the arcs have identifiers (see Figure 5.3). Let us avoid the use of *label* for identifier, to be able to use it later, but let it be clear that O is what is known in mathematics as a labeled, directed multigraph allowing multiple self-loops. O consists of a set of nodes D (identified by identifiers), a set of multi-arcs R, and a set of arcs A. Instead of speaking of tail nodes and head nodes for arcs, we will speak of subject nodes and object nodes, respectively. A multi-arc $r \subseteq A$ denotes

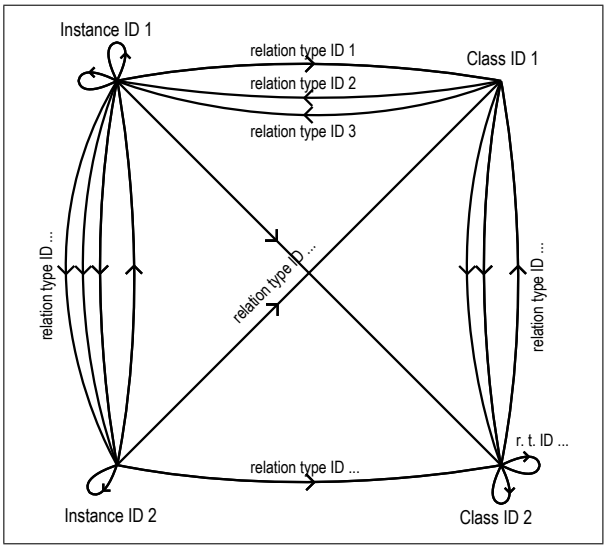


Figure 5.3: An MDG with 4 nodes, 12 multi-arcs (3 self-multi-arcs) and 18 arcs (5 self-arcs). For instance, the three arcs from instance 1 to instance 2 form a single multi-arc. Also single arcs are counted as one multi-arc. The multi-arcs represent relations (maximum 16 for 4 nodes) and the arcs represent instantiations of these relations in relation types (unlimited number).

the set of arcs between two nodes (from a subject node to an object node). Therefore, a multi-arc is identified by a pair of identifiers, consisting of a subject node identifier and an object node identifier. The cardinalities of R and D are related: $\#R \leq (\#D)^2$.

D consists of two disjoint sets of nodes, set I containing the instance nodes and set C containing the class nodes of the domain of discourse. The classes are assumed to have a definition outside the MDG. Also labels for the nodes and the set of relation types T are assumed to exist outside the MDG.

5.1.4 Relation arcs and relation multi-arcs

We can distinguish four important subtypes for relation arcs: instance relation arcs, class relation arcs, instance-class relation arcs and class-instance relation arcs, depending on the types of the nodes where the relation arcs start and end. An **instance relation arc** goes from an instance node to an

instance node, while an **instance-class relation arc** goes from an instance node to a class node. **Class relation arcs** and **class-instance relation arcs** are defined similarly. For relation multi-arcs we can make exactly the same distinction: instance relation multi-arcs, class relation multi-arcs, instance-class relation multi-arcs and class-instance relation multi-arcs.

- instance relation arc: Troy *is located in* Turkey.
- class relation arc: lion *eats* zebra.
- class-instance relation arc: human *lives on* Earth.
- instance-class relation arc: ISS *is residence of* astronaut.
- instance relation multi-arc: Julius Caesar *is born in* Roman empire and Julius Caesar *extends* Roman empire.

These examples are represented by their labels, but the actual relation arcs consist of identifiers like:

http://en.wikipedia.org/wiki/Troy
http://www.semantic-systems-biology.org/SSB#located_in
http://en.wikipedia.org/wiki/Turkey

5.1.5 Relation types

As every relation arc expresses a relation type that is unique within a relation multi-arc, it makes sense for relation types to define subtypes, supertypes and disjointness. A relation type t_1 is a subtype of a relation type t_2 (and t_2 a supertype of t_1) if all the relation multi-arcs that contain a relation arc declaring t_1, also contain a relation arc declaring t_2. Two relation types t_1 and t_2 are disjoint if there are no relation multi-arcs that contain two arcs where one expresses t_1 and the other t_2. These definitions imply a hierarchy of relation types, which is known in DL as a role hierarchy.

We can also distinguish four special classes of relation types (Relation Type Classes or RTCs): the **instance relation type**, the **class relation type**, the **instance-class relation type** and the **class-instance relation type**. A relation type t is an instance relation type if only instance relation arcs express t. Similar definitions apply for the other types.

Relation Type Class	Domain	Range
Class relation type	$Sub(c_d)$	$Sub(c_r)$
Instance relation type	$Ext(c_d)$	$Ext(c_r)$
Instance-class relation type	$Ext(c_d)$	$Sub(c_r)$
Class-instance relation type	$Sub(c_d)$	$Ext(c_r)$

Table 5.1: Domain and ranges for different types of relation types, assigned by two classes c_d and c_r.

Every relation type $t \in T$ may have a domain and a range assigned, defined by two well-chosen class nodes $c_d, c_r \in C$. If the domain and the range are set identical, they are defined by only one node $c_d \equiv c_r$. Let us call $Sub(c)$ the set of all the nodes in C that represent a subclass of a certain class $c \in C$ (also $c \in Sub(c)$). We call $Ext(c)$ the set of all the nodes in I that represent an instance of c. For the class relation type the domain is $Sub(c_d)$ and the range is $Sub(c_r)$, for the instance relation type the domain is $Ext(c_d)$ and the range is $Ext(c_r)$, for the instance-class relation type the domain is $Ext(c_d)$ and the range is $Sub(c_r)$ and for the class-instance relation type the domain is $Sub(c_d)$ and the range is $Ext(c_r)$ (see Table 5.1). The domain and the range of a relation type t should be chosen such that the subject node of any relation arc declaring t is always in the domain of t and the object node of that relation arc is always in the range of t.

In the triple representation of the MDG, the relation types can only occur in the central place of the triples.

5.1.6 The Metarel interpretation: multi-arcs as relations

A multi-arc is the set of all the relation arcs that start from a given subject and end in a given object. The (relation type) identifiers of the arcs within a multi-arc are unique. Therefore such identifiers can define types of multi-arcs. The multi-arc is an abstract concept that represents the relation from the subject to the object, in all its aspects. Relation types are like classes of multi-arcs that share the same type of relation arcs. We can call the multi-arcs *relation instances* or simply *relations*. Similarly as for natural language, relations are identified by a subject and an object. This is the interpretation that will be used for Metarel.

For instance, the mother-baby relation is a ‘feeds’ relation. But the same relation is also a ‘gives birth to’ relation. That is why the mother-

baby relation is better represented by a multi-arc than a single arc. The baby-mother relation is another relation which instantiates the relation type 'needs'.

5.1.7 Relation types as mathematical relations

Properties of a relation type t will be defined in this section by using some well-defined properties of an associated mathematical binary relation $\mathcal{R}$. The latter is defined by a set of elements $\mathcal{V}$ and a set of couples $\mathcal{E}$ in $\mathcal{V} \times \mathcal{V}$. For the mathematical interpretation of t, let $\mathcal{V}$ be the set of graph nodes D, and let $\mathcal{E}$ consist of the relation arcs declaring t. Now t is called transitive if $\mathcal{R}$ is transitive and t is called symmetric if $\mathcal{R}$ is symmetric. Two relation types t_1 and t_2 are called each other's inverses if their associated binary relations $\mathcal{R}_1$ and $\mathcal{R}_2$ are each other's inverses (also called converses).

The chosen domain and range of a relation type t do not always correspond with those of its associated binary relation $\mathcal{R}$. For a binary relation $\mathcal{R}$ the domain is defined as follows: an element x is in the domain of $\mathcal{R}$ if there exists an element y such that the couple (x, y) belongs to $\mathcal{R}$. This is stronger than the definition for the domain of a relation type t, for which only holds that if a subject s is not in the domain of t, then there will be no relation arc declaring t with subject s. Similar definitions apply to the range. We can still conclude that the domain of t must be chosen as a superset of the domain of $\mathcal{R}$ and the range of t as a superset of the range of $\mathcal{R}$.

The difference in definitions for domain and range accommodates for a fundamental difference between Mathematics and Ontology. Mathematics can assume perfect knowledge over the domain of discourse, being the elements and the set of couples. Ontology uses domains and ranges as semantical properties to make general statements about a domain of discourse which is known imperfectly, or too large to describe in all its detailed exceptions. Let us give an example: if the domain of a relation type 'has child' is set to 'living organism', it means that *only* living organisms *can* have children. It does not have to mean that *all* living organisms *actually do* have children.

Also for reflexivity of relation types another definition is needed than the definition that would follow from the associated binary relation. A relation type t is reflexive if the binary relation $\mathcal{R}_d$ is reflexive, where the domain of t is taken as the set of elements $\mathcal{V}_d$ for the binary relation $\mathcal{R}_d$. When an ontologist states that a relation type 'is located in' is reflexive,

he means that everything *that can be located in something* is located in itself. Without this special definition it would require that just everything is located in itself, even immaterial things (like functions or temporal intervals) that do not even have a location. The domain of a reflexive relation type necessarily equals the domain of its associated binary relation, because everything in the domain is at least related to itself.

5.1.8 Overview of the terminology

An overview of the terminology is given in Table 5.2. The use of 'relation' in Metarel is compared with RDF, OWL, DL and Mathematics. Arcs and multi-arc are syntactical artefacts, and are not entities that are itself described by Metarel. A multi-arc describes a relation, whereas an arc can be considered as an axiom that represents the fact that a certain relation is an instance of a certain relation type.

Apart from 'couple', which is interpreted as ordered pair in Mathematics, not many other terms are in use to denote a Metarel relation. Both 'couple' and 'pair', however, are terms that may be somewhat confusing in an ontological context. If they come in use in the field of Ontology, they may be confused with individuals that represent the mereological sum of two other individuals, or with a class of two individuals. Also in natural language, the couple John-Mary is viewed as something different than the relation that John has to Mary. Distinguishing relation from relation*ship* for solving this issue might create even more confusion. In spite of these considerations, the reader should be well aware that Metarel makes a controversial choice by calling 'relation' what is called 'couple' in Mathematics.

5.2 Formalizing logic and semantics through relations

The notion of 'relation' was defined within an MDG that has either classes or instances as nodes. Now a system will be proposed, consisting of three classifications, to provide a logical meaning to such relations: a classification of relations, a classification of relation types and a classification of relation type axioms. The union of the three classifications is a special type of relation ontology that will be called a *Metarel ontology*. It requires RDF graphs instead of MDGs for its representation. The goal of the Metarel ontology is to formalize the logical meaning of relation types. These will be used as subjects and objects inside the Metarel ontology.

(S,P,O)	Metarel:	relation arc
	RDF:	triple
	DL:	assertion
P	Metarel:	relation type
	RDF:	predicate
	OWL:	property
	DL:	role
	Mathematics:	relation
(S,O)	Metarel:	relation
	Mathematics:	couple

Table 5.2: The terminology in Metarel compared with terminology in RDF, OWL, DL and Mathematics. They are compared on the basis of a generic Subject-Predicate-Object structure, as used in RDF.

5.2.1 Metarelations

Obviously certain relation types are related to each other. For example, the relation type *has descendant* is a supertype of the relation type *has child*. The relations between relation types will be called 'metarelations'. The creation of metarelations requires the more general RDF graph instead of an MDG.

5.2.2 Labels for relation types

Let us analyze now the following two relation arcs, or triples, represented by their labels: '*Zeeland* is part of *The Netherlands*' and '*province* is part of *country*'. They seem to be using the same relation type *is part of*, but actually their relation types cannot be the same. The first 'is part of' must be an instance relation type, as it connects two instances, and the second 'is part of' must be a class relation type. Indeed we are missing some semantics in the sentence 'province is part of country'. What is meant would probably be something like: 'Every province is part of some country'.

Although both relation types may have the same label 'is part of', they are in fact different and they have a different identifier. It would often be hard to find human readable labels that can serve as identifiers in an RDF graph, as most quantifiers and modifiers of relations do not fit in a single triple. Sentences like '*province* all-some class relating part of *country*' are

not to be preferred. For the virtues of user-friendly browsing, visualizing and text searching, it is better to hold an intuitive and consistent representation with labels. A good rule for the creation of such labels is to use a verb in them, conjugated in the third person singular. Such labels may not express all the intended meaning but they lay a basis for natural language representation. The full semantics will be hidden in the identifiers of the relation types, which are formalized in the Metarel ontology. The identifiers can be used by visualization systems, query systems, reasoners and for Knowledge Management purposes. Natural language processing systems can use the semantics of the identifiers to conjugate the verbs in the labels properly and add quantifiers like 'every', 'a', 'some', *etc.*

5.2.3 A classification of relations

As was discussed before, relation types can be ordered in a hierarchy, which has relation types as classes and relation multi-arcs (or simply relations) as instances. It corresponds to the role hierarchy in DL. This classification is important for computational reasoners, as it allows to derive supertypes from subtypes.

The assertion that a relation is an instance of a relation type, does not belong to the Metarel ontology. This is exactly what a relation arc asserts in the MDG. Only the subsumption of a relation type by another relation type needs to be asserted in the Metarel ontology.

An example is the relation between a protein type PAF1 and DNA recombination. If this relation is classified as a 'negatively regulates', then it is automatically also classified as a 'regulates'.

5.2.4 A classification of relation types

The classification of relation types is a metaclassification compared to the classification of relations. An example is the classification of 'is part of', 'is located in' and 'is preceded by' as transitive relation types. Relation types are instances in this classification. Relation types with common semantic features (like reflexivity or transitivity) can be grouped in classes of relation types (like 'reflexive relation type' and 'transitive relation type'). Such classes will be called *relation type classes* (RTC). The whole classification is shown in Figure 5.6. Multiple inheritance allows that a certain relation type instantiates several RTCs. The RTCs 'instance relation type', 'class relation type', 'instance-class relation type', *etc.* have been discussed

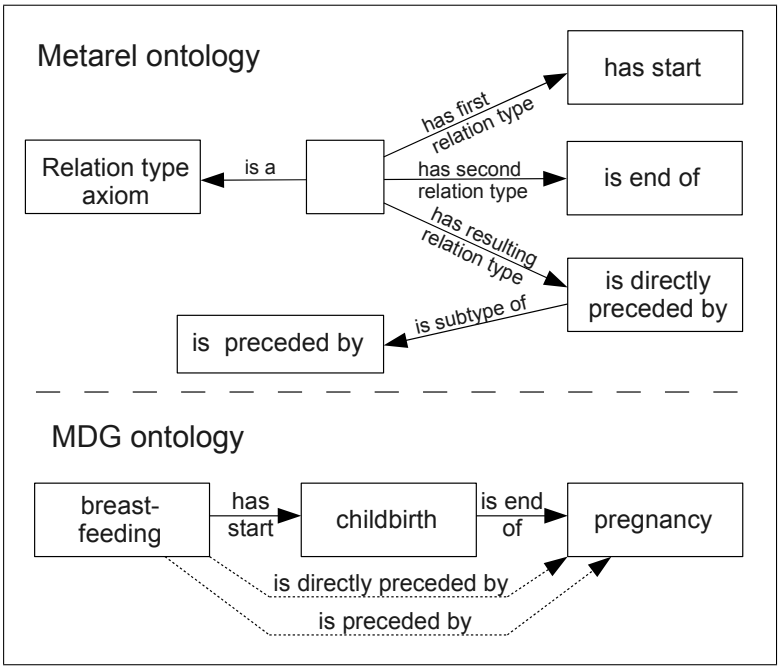


Figure 5.4: Two relation arcs 'is directly preceded by' and 'is preceded by' can be derived in the MDG from the classification of relations and relation type axioms in the Metarel ontology.

before. They can be classified as pairwise disjoint here.

An important subtype of relation types are those that are based on an underlying instance relation type. Such a relation type will be called an 'instance-based relation type'. Most instance-based relation types will be instance-class relation types, class-instance relation types or class relation types, but it could also be a relation type from a more generic RTC. Metarel acknowledges that other relation types can exist between classes, instances and classes and between concepts in general. Classifying a class relation type like 'has a longer class name than' would not be an instance-based relation type. For allowing extensions 'instance-based relation type' is added as a sibling of the other metaclasses in the hierarchy, assuming multiple inheritance between all of them. This means that any relation type can always be classified as both an instance-based relation type and for example a class relation type.

5.2.5 A classification of relation type axioms

The semantics of a relation type is defined by relation type axioms. Some axioms apply to a single relation type (for instance '*is part of* is reflexive'). These axioms can be asserted by creating an appropriate RTC and classifying the relation type as one of its instances. Other axioms apply to two relation types (for instance '*is part of* is the inverse relation type of *has part*'). These axioms can be asserted with a metarelation between the relation types. However, there are also axioms that apply to three different relation types, in particular for relation types that form chains. For instance the relation between two classes A and C is an *is directly preceded by*, whenever the relation between A and a class B is a *has start* and the relation between B and C is an *is end of*.

In a graph-based representation format, such axioms require a proper, identifiable node in the Metarel ontology, which can be connected with the relation types through metarelations. These metarelations can suffice to provide the intended meaning to the axiom (see Figure 5.4).

5.3 Metarel in the Semantic Web

5.3.1 Internationalized Resource Identifiers for Metarel

In order to be useful for the Semantic Web, the terms in the Metarel vocabulary need IRIs as identifiers. Because the vocabulary will be used for the management of RDF stores, the IRIs would better be human readable. They will show up everywhere in scripts and update queries that are dealing with the management of the store. Also OWL and RDF(S) use human readable IRIs for their vocabulary.

The terms in the Metarel vocabulary will get an identifier in the namespace of *http://www.metarel.org/*, such as *http://www.metarel.org/RelationType*, abbreviated *metarel:RelationType*. The use of *http*-style IRIs is a recommended practice for Linked Data on the Semantic Web. Any web agent can easily verify that the namespace *http://www.metarel.org/* is in use by trying to resolve any IRI in this namespace via Domain Name Servers. This gives a guarantee that any IRI created in this namespace is unique worldwide.

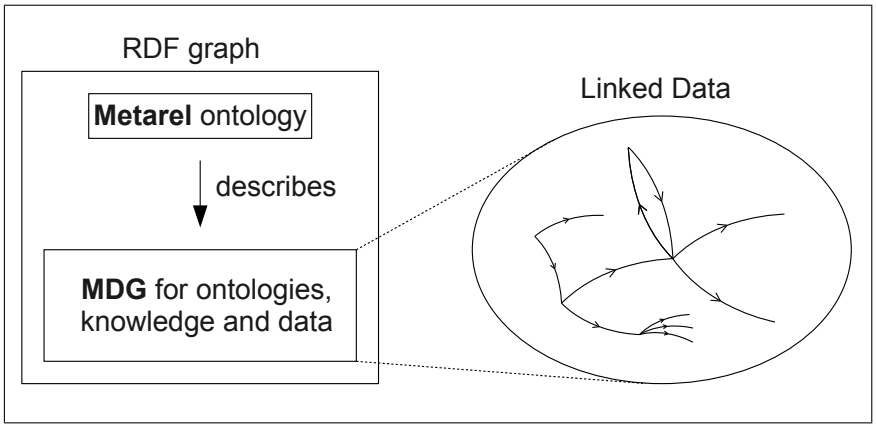


Figure 5.5: Metarel can be loaded in any RDF graph and it can be used to describe the meaning of those triples that can be interpreted as an arc that is logically meaningful within an MDG that separates classes from instances.

5.3.2 Practical usage

Metarel is released in an RDF format, metarel.rdf[1], which can be loaded in OWL software systems as valid OWL Full, and as an RDF graph in RDF stores. The Metarel ontology can be exploited by adding it to the same RDF graph or RDF store that contain Metarel-described relation types. Query and rule libraries, like libraries of SPARQL and SPARQL/Update queries, can address both the Metarel ontology and the RDF data. The usage of metarel.rdf is depicted in Figure 5.5 and will be demonstrated in Chapter 7.

5.3.3 Engineering Metarel with existing vocabulary

We will now analyze which terms in Metarel equate with corresponding terms in OWL. Since the release of OWL 2, W3C's most advanced ontology language consists of different DL-based profiles, like OWL DL and OWL EL, on the one hand, and an unrestricted combination of all the language constructs, OWL Full, on the other hand. They all share the same language constructs, identified with exactly the same URIs, however, their semantics depends on the context in which they are used. OWL profiles use the Direct Semantics (DS) [133], whereas OWL Full assumes the RDF-Based

[1] http://www.semantic-systems-biology.org/inhouse/metarel.rdf

Semantics (RBS) [134]. This practice has kept the number of metaconcepts in OWL relatively low so far, although their meaning remains dependent on the context.

The RDF-Based Semantics suits OWL Full, which was created to support many database and knowledge representation systems [108]. This semantics allows RDF tool builders to implement support for important metaconcepts like *owl:inverseOf*, *owl:sameAs*, *owl:TransitiveProperty*, *etc.*, within RDF stores. In order to profit from this support and in order to allow compatible support for class relation types, the use of the RDF-Based Semantics is required. However, the meaning of some metaconcepts in Metarel will be clarified by equating them with metaconcepts under the Direct Semantics. In case the Direct Semantics is meant in the ensuing explanation, the URIs will be extended with -DS. Otherwise, the RDF-Based Semantics is assumed. All the metaconcepts can still be merged in a single hierarchy, as is shown in Figure 5.6.

The most central RTC in OWL, the root of the metaclassification that parallels 'relation type', is identified as *rdf:Property*. It has the same meaning in both OWL-RBS and OWL-DS. Its description as "a relation between subject resources and object resources" stems clearly from the efforts to create compatibility with RDF, where everything is considered as a 'resource' [107]. But within the OWL documentations, the word 'property' is systematically considered as a relation between instances (also called individuals). Also from the definitions of all the metaconcepts that have a place lower in the hierarchy, like *owl:ObjectProperty*, *owl:TransitiveProperty, etc.,* it follows that properties stand only between instances. This means that while RDF considers everything (instances, classes, relations, data, websites) as a resource, OWL-RBS considers all these things as *instances*.

The root of the hierarchy of the relation type classification, *metarel: RelationType*, can be equated with the root of the property model in OWL Full, *rdf:Property*. Below that come *owl:DataTypeProperty*, *owl:AnnotationProperty* and *owl:ObjectProperty* in the hierarchy of OWL Full. All the relation types that were discussed for Metarel are object properties since *owl:ObjectProperty* is formally equivalent to *rdf:Property* [167]. *metarel: ClassRelationType* and *metarel:InstanceRelationType* come below *owl:ObjectProperty* in the hierarchy, as disjoint classes, although neither can equate with anything in OWL-RBS. *metarel:TransitiveRelationType* can equate with *owl:TransitiveProperty* and *metarel:SymmetricRelationType* with *owl: SymmetricProperty*. *rdfs:subPropertyOf* can be used between a relation

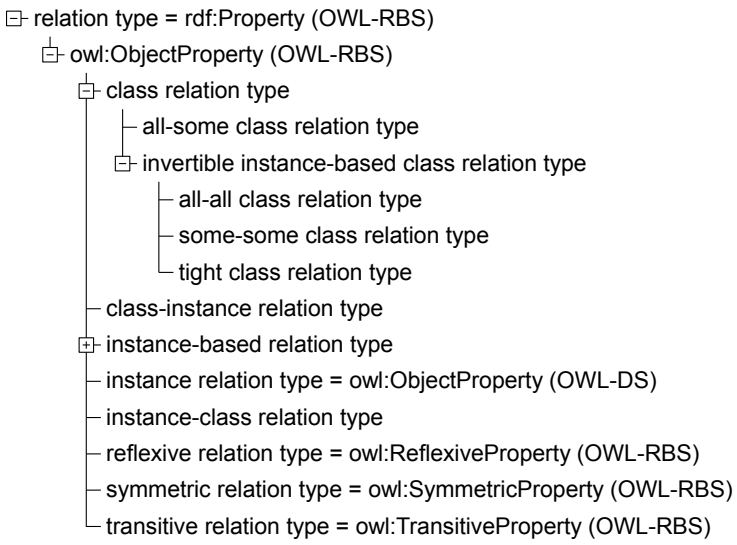


Figure 5.6: The relation type classification integrated with the RDF-Based Semantics (OWL-RBS) of the OWL property model. The instance relation type equates with the Direct Semantics (OWL-DS) for *owl:ObjectProperty*. The classification assumes multiple inheritance instead of disjointness. For instance, some class relation types may be transitive relation types, others may not be.

type and its supertype and *owl:inverseOf* can be used between any relation types that are each other's inverses.

The metarelations *rdfs:domain* and *rdfs:range* for domains and ranges have a more narrow definition than *metarel:hasDomain* and *metarel:hasRange*. Following the RDFS and OWL documentation, properties that are assigned a domain, resp. range, cannot have classes as subject, resp. object [107, 108]. This excludes class relation types from having domains and ranges in OWL-RBS. However, these OWL constructs can be used for the instance relation types.

These are the most important terms in OWL that equate with Metarel terms in the RDF-based semantics, but some more terms equate within the direct semantics of OWL. The most central of all is the notion of 'instance' in Metarel, which can be equated with *owl:Individual*-DS. This implies a commitment to the same distinction between instances, classes, data and annotations as for OWL-DS. In this semantics *owl:ObjectProperty*-DS

can be equated with *metarel:InstanceRelationType*. *Metarel:isInstanceOf* equates with *rdf:Type*-DS and *metarel:isSubClassOf* with *rdfs:subClassOf*-DS. These two relation types might be classified as instances of respectively *metarel:InstanceClassRelationType* and *metarel:ClassRelationType*. Properties like *owl:sameAs*-DS and *owl:differentFrom*-DS can be classified as instances of *metarel:InstanceRelationType*. The OWL-DS meaning of the metaconcepts that come below *owl:ObjectProperty*-DS can be formed through multiple inheritance in Metarel. *Owl:TransitiveProperty*-DS, *owl:SymmetricProperty*-DS and *owl:ReflexiveProperty*-DS correspond to transitive instance relation type, symmetric instance relation type and reflexive instance relation type, respectively.

Although RTCs like *metarel:ClassRelationType*, *metarel:InstanceClassRelationType* and *metarel:ClassInstanceRelationType* still cannot equate with any particular OWL language construct in the Direct Semantics, they can be represented by a composition of several OWL constructs (as shown in Figure 5.7 B). They are called property restrictions by value constraint in OWL. The different types of class relation types (all-some, all-all, some-some and tight) are all property restrictions with existential quantification. The virtue of Metarel is exactly to have the composed representation of such property restrictions compacted into a single triple.

By this equalization of the Metarel vocabulary with OWL, it becomes possible to use Metarel in RDF together with (any flavor of) OWL ontologies in an RDF store and have compatible support for the whole store. It enables the development of reasoners that operate on this union and tools that create conversions between both types of representation.

5.3.4 Relational vocabulary beyond OWL

The virtue of Metarel is the use of instance based relation types, modeled as a single triple. These are the relation types for which OWL requires a complex of several triples.

Probably the most important instance-based relation type is the 'all-some class relation type'. If the class relation type 'is part of' in the triple 'province is part of country', is classified as an all-some class relation type, it means that 'all provinces are part of some country'. A metarelation is used to express that the class relation type 'is part of' *'is based on'* the instance relation type 'is part of'.

The inverse of an all-some class relation type is not another all-some class relation type. It requires another metarelation to relate two all-some

class relation types that are both based on a pair of inverse instance relation types. Such class relation types are each other's *reciprocals* [38]. Class relation types that are each other's inverses and that are based on a pair of inverse instance relation types are *invertible instance-based class relation types*. The *all-all class relation type*, the *some-some class relation type* and the *tight class relation type* are distinguished depending on the quantification method that is used to base them on instance relation types.

The all-all class relation type is based on an instance relation type in such a way that every instance of the subject class has a relation arc, declaring the instance relation type, to every instance of the object class. This means that every relation arc declaring an all-all class relation type corresponds in mathematics to a complete relation between two classes. For instance, the all-all class relation type 'knows', based on the instance relation type 'knows', applies between globetrotter and continent. Every globetrotter knows all the continents. The RTC of some-some class relation types is defined similarly as those for the all-some and all-all class relation types. 'Human eats animal' will imply that some humans eat some animals, if this relation type is asserted to be a some-some class relation type. A double implication follows from asserting that a relation type is a tight class relation type. An ontology engineer created a relation type 'is located in' and asserts it as a tight class relation type, based on the instance relation type 'is located in'. Then from 'city house is located in city' follows that every city house is located in some city and that for every city, there is some city house that is located in this city. The relation type 'is integral part of' in RO, the relation ontology the OBO Foundry, was defined as a tight class relation type, based on the instance relation type 'is part of'.

The distinction of RTCs of relation types here is not intended to be exhaustive. Class relation types were also investigated in the Class-Relationship logic of J.F. Nilsson [168] and as regulatory relations between classes of molecular entities by S. Zambach [169]. Metarel is however not a logic, but is provided as a vocabulary that allows to implement different logics by defining exactly which kind of RTCs and which semantic rules are used in that language.

5.4 Discussion

Many researchers involved in ontology engineering are only acquainted with OWL, the official ontology language of the Semantic Web. For this

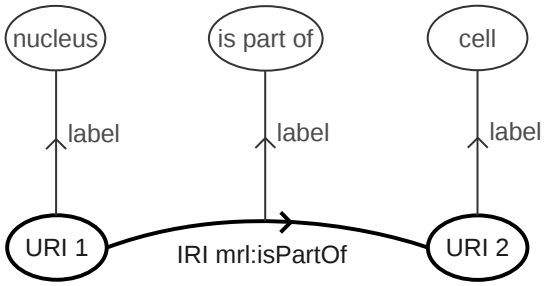
reason it was important to make a correspondence between the vocabulary for relations used in Metarel and the relational vocabulary used for OWL.

OWL uses RDF as a syntax, without restricting to the use of MDGs. The vocabulary for OWL will thus necessarily deviate from the Metarel vocabulary, which describes the logical meaning of relations in MDGs. Instead of describing RDF relations directly, OWL uses RDF as a syntax to describe DL-based relations. This has resulted in OWL representations that are hard to comprehend at the level of RDF. Because DL bases every relation on relations between instances, only the instance relation arcs are represented with a single triple in RDF. Other cases, like class relation arcs, require a complex with a blank node in the OWL/RDF syntax, as can be seen in Figure 5.7. In spite of this difference, there is the important similarity that the central OWL concept 'object property' corresponds to the 'instance relation type' in Metarel.

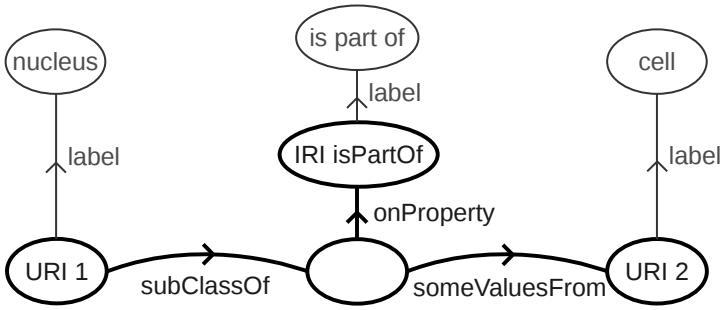
In fact, OWL does not use any word that is related to 'relation', instead it uses 'property'. For a knowledge statement in the triple form, like 'Lea *is daughter of* Mike', the OWL vocabulary, as it is used in the OWL documentation, would say that Lea has the *is daughter of*-property with value 'Mike'. The usage of 'property' is fairly natural for properties that are functional, i.e. properties that can have only a single value: my teacup has the *has color*-property with value 'white', it has the *has weight*-property with value '0.2 kg' *etc.* This usage is much less natural for properties that can have many values, for instance: Lea has the *has friend*-property with value 'John'. This sounds as if Lea has only a single friend. Moreover, in the natural language usage of property, we rather think of a property as the combination of the OWL property and the value: 'having a white color' would be one of the properties of my teacup. For this reason, the relation-centric vocabulary of Metarel can be useful to clarify the meaning of some terms in OWL.

RDF is probably the most popular Semantic Web technology. It is used for large data stores and many classical databases in the Life Sciences have converted their contents into RDF. The main reason for its success seems to be the ease of modeling and querying in RDF. Its intuitive semantics, which interprets a triple as a sentence with a subject, a predicate and an object, allows for an intuitive querying in SPARQL that does not require any knowledge of logic.

There exists another Semantic Web vocabulary, apart from the OWL vocabulary, for describing the logical meaning of RDF graphs, namely



A. Direct arc in METAREL



B. Indirect complex in OWL/RDF

Figure 5.7: A relation in RDF described by Metarel compared to a complex representation with explicitly modeled quantifiers in the OWL/RDF syntax. All the information in the complex is also expressed by classifying the URI mrl:isPartOf as an all-some relation type in a Metarel ontology that belongs to the same RDF graph.

RDF(S). Some part of the vocabulary in RDF(S) was even reused in OWL. Also for RDF(S), some correspondences with Metarel can be identified. However, some of the terms that are needed in Metarel do not exist in RDF(S), nor in OWL.

Also the approach of RDF(S) differs, mainly because it does not make a distinction between instances and classes. RDF(S) interprets the nodes in RDF as *resources*. Therefore RDF(S) does not provide a rich vocabulary for describing the semantics of logic-based relations.

Knowledge engineers who are in doubt whether to use OWL, based

on DL, or rather MDGs described by Metarel, should weigh two aspects: expressivity versus queryability. In addition to being less standardized, MDGs plus Metarel is also less expressive from a logic point of view. On the other hand, the paradigm subscribed by Metarel guarantees easier querying through SPARQL, which may be important for many practically oriented use cases.

5.5 Conclusion

Metarel was presented as a Semantic Web vocabulary for ontologies that consist mostly of relations between classes. A Metarel framework consists of an ontology represented as an MDG (multidigraph) in RDF and a relation ontology (the Metarel ontology). Every statement in the MDG is modeled as exactly one RDF triple consisting of a subject, a predicate and an object. This direct, short syntax for knowledge statements has a lot of potential for Semantic Web representations.

Metarel is based on a formal analysis of relations between instances and classes, which is compared with relations in mathematics. It distinguishes relations from relation types and relation arcs. Relation types correspond to properties in OWL and the relation arcs are the RDF triples. Relations are instances of relation types.

SPARQL querying on MDGs is much shorter, intuitive and efficient compared to SPARQL querying on OWL/RDF graphs. Apart from a shorter RDF syntax for relations between classes, Metarel is maximally compatible with OWL and can easily be translated with Semantic Web tools for usage in the existing OWL profiles.

RDF triples appeal to natural language. A consistent manner of assigning labels to relation types was proposed in order to allow natural language processing systems to build sound sentences from triples. Metarel can lower the threshold for conversions of RDF triples towards more expressive logic languages like OWL.

Chapter 6

BioGateway: a Semantic Web Knowledge Base

6.1 Introduction

The practical experiences from developing CCO and the DIAMONDS platform in RDF, as explained in Chapter 4, and the theoretical ideas for modeling RDF with Metarel, discussed in Chapter 5, form a good basis for the engineering of a knowledge base that brings the theory a step further into the practice. In this chapter, BioGateway will be introduced, which is the successor of the DIAMONDS platform.

BioGateway is a protein-centric RDF knowledge base that is oriented towards genome researchers with questions about genes and proteins, their functions, interactions, cellular locations and their involvement in cellular processes and diseases. It allows to generate new hypotheses through advanced queries within this knowledge area, which might be the input for further experiments in biotechnological laboratories. BioGateway was first presented at the Semantic Web Applications and Tools for Life Sciences (SWAT4LS) conference in Edinburgh in November 2008 and was published in BMC Bioinformatics in October 2009.

Whereas CCO is a knowledge base that integrates knowledge related to a specific research topic, namely the cell cycle, and is restricted to only four species, the size of BioGateway is much larger. It is an RDF knowledge base that integrates all the OBO Foundry candidate ontologies, the Gene Ontology Annotation files (GOA) with genomics data for about 2000 species, the SWISS-PROT protein set (the best curated part of UniProt) and

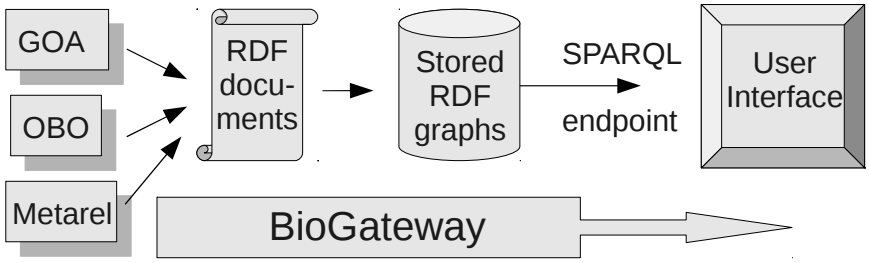


Figure 6.1: The (bio-)gateway from original knowledge sources towards the end user.

the NCBI taxonomy, containing information about thousands of species in the tree of life [170]. These original resources, that were already included partially in CCO, are also translated to RDF with ONTO-PERL [1]. Also Metarel is integrated in BioGateway, for the management of relations and relation types, as well as MetaOnto, with information about the ontologies and the RDF graphs themselves. MetaOnto was specifically developed for BioGateway and will be explained in more detail later. All these resources are loaded and queried in a Virtuoso system [153]. The generic architecture of BioGateway is shown in Figure 6.1.

Except for UniProt, no restrictions were applied to reduce the size of the knowledge base by loading only a part of the available data. All the original sources are available either directly through a SPARQL endpoint[1], or indirectly through a web interface that queries this endpoint. Two web interfaces are available: the SPARQL query interface[2] with preconstructed queries that can be parameterized and the SPARQL-viewer[3] with a visual display of query results.

The projects BioGateway, CCO, ONTO-PERL and Metarel were identified as belonging to a larger research domain that can be called *Semantic Systems Biology* (SSB). This domain is an extension of the interdisciplinary research domain of Systems Biology, defined by Hiroaki Kitano as a cooperative cycle of research efforts in computer science, mathematics and biology [171]. SSB extends this cycle with a semantics component by adding the research efforts in knowledge management and logic. This

[1] http://www.semantic-systems-biology.org/biogateway/endpoint
[2] http://www.semantic-systems-biology.org/biogateway/querying
[3] http://www.semantic-systems-biology.org/biogateway/sparql-viewer

idea was first expressed together with the first version of BioGateway in a paper that was presented for the SWAT4LS conference in Edinburgh in 2008 [6]. It was affirmed in the International Conference on Web Intelligence in Sogndal in 2011 [172].

All the projects falling under the umbrella of SSB were grouped on the same website[4], hosted by the NTNU in Trondheim.

6.2 Cyclic development of RDF models

A suitable data representation is necessary to share and uniformly query the RDF-integrated repository. Although different means exist to translate ontologies in the Open Biomedical Ontologies Format (OBOF) into several formats [173], there is no accepted mapping of OBO ontologies to RDF. The development of BioGateway, which includes all the OBO ontologies in RDF, implies the development of a mapping from OBO to RDF. This mapping was devised to retain a high-fidelity conversion (*i.e.* no information is lost) and facilitate querying. It extends and improves the mapping from the RDF export function in the ONTO-PERL suite. A very important method to achieve a high quality for the mapping has been the feedback through querying of any new mapping proposals. This has resulted in a cyclic development of the mapping to RDF.

The proposed mapping from OBO to RDF has undergone several refinements not only to capture all OBO specification elements [174], but also to have a relatively natural translation, allowing users familiar with the OBO format to immediately recognize the corresponding tags. More details about the proposed format conversions can be found within the ONTO-PERL source code [175], and the entire list of RDF-ied resources that are integrated into BioGateway can be found on the project download web page[5].

Before any new resources were integrated in BioGateway, a library with some SPARQL queries was created. These queries could test whether any newly integrated RDF triples were functioning syntactically and semantically as expected. This library was used already for querying and browsing through CCO in the DIAMONDS platform. The following queries were retained for testing BioGateway:

[4] http://www.semantic-systems-biology.org

[5] http://www.semantic-systems-biology.org/biogateway/download

```
get_children
get_comment
get_definition
get_heads_by_relation_type
get_hierarchy_by_relation_type
get_hierarchy_to_root
get_name
get_neighborhood
get_parents
get_properties
get_relation_types_of_ontology
get_relations_to_selection
get_root_of_ontology
get_subnamespace
get_subnamespaces_of_ontology
get_synonyms
get_tails_by_relation_type
get_type_of_uri
get_xrefs
get_1_step_interaction_path
get_2_step_interaction_path
get_3_step_interaction_path
search_terms_on_name
search_terms_on_properties
```

There are different reasons why feedback through querying is necessary during the construction of an RDF knowledge base. Here are some of the important feedback practices:

- **Test any new mappings on obvious errors.**
  The RDF syntax validation is obviously a primordial condition for loading any RDF document in a triple store. But querying through SPARQL is the only way of testing the usefulness of an RDF knowledge base.

- **Present human readable answers on queries.**
  As RDF works with IRIs, many outputs from random SPARQL queries might be difficult to comprehend. The queries in the library allow for a clear presentation by using the most comprehensible labels of the terms in the knowledge base.

- **Reduce unnecessary chains and blank nodes.**
  The XML syntax of RDF invites the creation of many unnecessary levels that translate into blank nodes and intermediate chains in the

graph. The RDF document may look nice whereas the underlying graph model is very awkward. It is very hard to predict the structure of the RDF graph model from within a programming environment that produces lines in the RDF/XML syntax. This graph model becomes immediately clear through SPARQL querying.

- **Verify that the recommended practices enable the basic queries.**
  It may appear that certain basic queries are possible through cross-references or through labels, but in practice this may turn out to be infeasible. Sticking to some recommended Semantic Web practices, like using the IRI's everywhere, will often suffice. On the other hand, certain practices only serve advanced automated reasoning, like those for OWL DL. These may form an obstacle for querying RDF with SPARQL.

  The RDF export tool of ONTO-PERL did not produce a single unique identifier per term in the first phases of the development cycle. Instead, it generated identifiers that were dependent either on the graph in which the resources were loaded, or on the source file from which they were derived. Because of that, a single identifier space was chosen for all the terms in BioGateway, independent of their origin. Such a space for http-style IRI's, http://www.semantic-systems-biology.org/, was created for the SSB project and it could be used for BioGateway. Any unnecessary additions to this prefixed string, that would only relate to specific files or graphs instead of universal terms, were avoided. This practice has established an efficient system for identifying the terms in BioGateway, independent of where the terms are used and where they are first described.

- **Establish RDF models that enable more basic queries.**
  Some queries are only enabled by restricting on the unbounded flexibility of RDF. For instance, the requirement that every term has exactly one label (which may thus be called a mandatory, preferred label), enables a human readable output for the queries. If some terms have no labels, or more than one label, the answers to queries may be incomplete or show up more than once. Many queries also require a systematic way to distinguish RDF resources that represent ontology terms from those that represent other things, like relations or metadata.

- **Avoid that the addition of new triples disables other basic queries.** Querying through SPARQL is essentially the reduction of the whole set of RDF resources in the knowledge base to a much smaller set. Therefore the inclusion of new sources to the knowledge base may destroy the functioning of queries that were established previously, because they can make the answer set unreasonable large or corrupt the set with results of poor quality, results that are not human readable or even false results. For instance, if a newly integrated source contains proteins that were already present in the knowledge base, but they contain multiple labels instead of exactly one, then all the queries that assumed a bijection between proteins and their labels will return every protein multiple times in the answer set. Even the addition of a single triple may corrupt many queries, for example when both the labels *is a* and *is_a* would be present for the subsumption relation type.

  Another example is the addition of logically inferred triples to the RDF graph with the original triples. Queries that gave nice results in the original graph may start to produce masses of uninteresting results. This has led to the design practice that RDF documents should be uploaded into two different graphs in BioGateway: a graph containing only the original triples, and a graph containing both the original and the inferred triples. The latter was suffixed with *_tc*, initially an abbreviation of 'transitive closure' and later renamed to 'total closure'.

- **Keep querying scalable during the growth of the knowledge base.** This practice differs from the previous ones. When many triples are added to an RDF graph, even if they keep the RDF model intact, certain queries that gave a quick response may suddenly start to slow down dramatically. The SPARQL engine has to search longer before it finds the correct matching of the triple patterns. This observation was the reason to have the knowledge base partitioned in even more separate graphs. This is especially necessary for the parts of the knowledge base that are heavily queried, like the Gene Ontology and the relation types.

- **Allow for a short syntactical expression in SPARQL for interesting queries**
  This aspect conflicts with the previous one, because integrated que-

ries will expect from the query person that he knows which triples are stored in which graph. If the query ranges over many graphs, for instance hundreds of species-specific graphs, he would have to write hundreds of lines. This problem could be solved by the use of redundancy in BioGateway. All the RDF documents are loaded in separate small graphs, in graphs of intermediate size and also in one big graph, called SSB. Depending on the type of query, either the small graphs or the bigger graphs can be addressed.

- **Establish a library of speedy, basic queries for testing.**
  There are many basic queries, like retrieving labels, subclasses or the neighborhood of a node, which need to operate correctly and quickly in order to allow even more advanced queries. As the knowledge base grows and becomes more advanced, such basic queries will often break. Every query ability that was enabled for the knowledge base needs to be represented by a query in the library that serves for testing any modification during the construction.

- **Establish a library of advanced queries that tackle specific research questions.**
  This is an important achievement for a knowledge base that is built for hypothesis generation by biological expert users. Such a library may be the starting point to parameterize, combine or extend the queries, depending on their outcome.

- **Enable quick answers on queries for hypothesis generation.**
  A relatively quick query answer is always a desirable feature for any system. A query builder wants to see a quick and sound answer to a small query pattern or on a small part of the data before he launches a heavier query. Therefore, the response time was systematically tested with the library of queries. This quality constraint turned out to be the biggest challenge during the development of BioGateway. The query performance is dependent on different factors, making the subject difficult to investigate. In particular, disk access and the caching of earlier results, as well as seemingly unimportant details in the query itself can have a profound impact on the query performance, resulting in differences of one to two orders of magnitude in query time. Due to these optimizations, most queries in the library return an answer within one second, and the chances of getting an answer to a more complex query within a reasonable time are also better.

- **Identify which other sources should be included to tackle a certain research question.**
  This aspect requires biological expertise in the first place, and the knowledge of other biological data sources. But it also requires experimentation through SPARQL querying in order to see whether the desired research question can actually be tackled. For instance, the retrieval of all the proteins involved in a certain (generic) type of disease was not possible in SWISS-PROT, because the diseases are only mentioned in a disease description instead of having a dedicated node in an ontological hierarchy. This could enable only literal searches on a specific disease. Also the inclusion of biological pathway data was identified as a desired asset when the retrieval of interaction chains by the data in SWISS-PROT did not appear to be very obvious.

The quality improvement is a task that can be extended endlessly. The numerous and diversified improvements that the feedback from querying brings for the quality of an integrated knowledge base underline the difficulties that will emerge in the querying of distributed RDF resources that were never engineered to be queried together. By the feedback process, BioGateway has reached a better quality compared to RDF stores that provide an unaltered aggregation of diverse resources.

6.3 The architecture of BioGateway

6.3.1 The identifier space

The feedback from queries resulted in many important architectural decisions for BioGateway. First of all, it resulted in a very manageable treatment of the identifiers. All the imported data sources, when converted into RDF graphs, share a basic IRI:

```
http://www.semantic-systems-biology.org
```

This means that each resource (*e.g.* each protein from SWISS-PROT, each taxon from the NCBI taxonomy, each OBO term) has an IRI of the form:

```
http://www.semantic-systems-biology.org/SSB#resource
```

Each of the imported data sources is represented as an individual graph with a specific IRI, of the following form:

```
http://www.semantic-systems-biology.org/graph_name
```

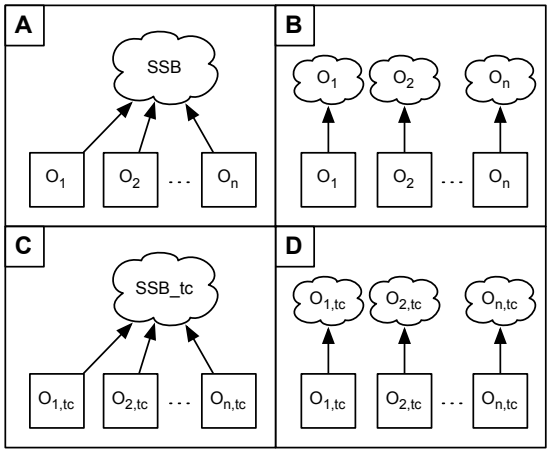


Figure 6.2: The architecture of BioGateway. The RDF source files (square boxes) are redundantly loaded into different RDF graphs (clouds). Every file goes into the big SSB graph (A) and into a separate small graph (B). SSB_tc and small graphs with the _tc-suffix contain logically inferred triples (C and D).

6.3.2 The division in graphs

An important architectural feature of BioGateway is its division in different RDF graphs, which can be seen in Figure 6.2.

The loading into graphs with different sizes is inspired by the practical experience of keeping the queries scalable during the growth of the knowledge base and allowing for a short syntactical expression in SPARQL for interesting queries. Apart from the big SSB graph, containing all the original RDF files, also two intermediate graphs were created: the OBO graph and the GOA graph, containing all the OBO formatted files and the GOA files respectively. The GOA graph became obsoleted later because the GOA files account for up to about 80 percent of the data in SSB. Using the SSB graph instead of the GOA graph for the same queries did not really perform less well, since the sizes are similar.

The separation of the graphs with logically inferred triples from the graphs with the original triples is inspired by the practice of avoiding that the addition of new triples disables other basic queries. Even though ontologies can be considered as logical frameworks, there are many queries

that may relate to the original hierarchy that was deliberately engineered by ontologists. Also in the data sources that consist of annotations, the original triples are more accurate and better reviewed compared to those that were inferred by automated logic techniques.

When inferred triples are loaded in the same graphs as the original triples, it becomes impossible to make queries that target the original triples exclusively. One of the ontological queries in the library is designed to find the closest common ancestor in the hierarchy of an ontology for two given terms. The idea of a closest common ancestor can best be understood within the original ontological hierarchy. Indeed, within a framework that contains all the triples that were inferred through a transitive closure, all the terms have the root of the ontology as a direct parent. This is the complete query:

```
# NAME     : get_common_ancestor
# PARAMETER: GO_0002617: the first query-term
# PARAMETER: GO_0034125: the second query-term
# FUNCTION : returns the closest common ancestor-term in the
#            hierarchy for two given terms
BASE   <http://www.semantic-systems-biology.org/>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX ssb:<http://www.semantic-systems-biology.org/SSB#>
PREFIX term1_id: <SSB#GO_0002617>
PREFIX term2_id: <SSB#GO_0034125>
SELECT distinct ?common_ancestor ?common_ancestor_id
WHERE {
  GRAPH <SSB_tc> {
    term1_id: ssb:is_a ?common_ancestor_id.
    term2_id: ssb:is_a ?common_ancestor_id.
    OPTIONAL {
      term1_id: ssb:is_a ?direct_child.
      term2_id: ssb:is_a ?direct_child.
      GRAPH <SSB> {
        ?direct_child ssb:is_a ?common_ancestor_id.
      }
    }
    ?common_ancestor_id rdfs:label ?common_ancestor.
  }
  FILTER(!bound(?direct_child))
}
```

For this query we need both the regular RDF ontology in SSB and its transitive closure in SSB_tc. In fact, the query might be reduced to: *find all the ancestors of both terms that do not have any descendants that are ancestral to both terms*. To find all the terms that are ancestors of both

terms, we need the transitive closure graph, as in that form all the ancestors are directly linked to their descendants. Two triples in the query are enough to retrieve their id:

```
GRAPH <SSB_tc> {
        term1_id: ssb:is_a ?common_ancestor_id.
        term2_id: ssb:is_a ?common_ancestor_id.
}
```

We get all common ancestors with this query, while we only want the closest ones. Therefore, we check for the children of this set of ancestors. This can be best accomplished in the ontology without transitive closure:

```
GRAPH <SSB> {
        ?direct_child ssb:is_a ?common_ancestor_id.
}
```

Additionally, we check whether these children belong to the same set of common ancestors as defined before:

```
        term1_id: ssb:is_a ?direct_child.
        term2_id: ssb:is_a ?direct_child.
```

The last two checks go in an optional clause, because we only want the common ancestors for which these checks fail. In this way, we can filter the common ancestors for which this kind of *?direct_child* does not exist:

```
        FILTER(!bound(?direct_child))
```

The query shows that it is necessary to keep separate graphs for the original triples and for the triples that were logically inferred from the original triples.

6.4 Metarel for relation management

6.4.1 The Relationship Ontology in the OBO Foundry

The framework for relations that was developed in Metarel was of great use for facilitating the relational management in BioGateway. Just like other translation tools, ONTO-PERL was designed for translating the relation types (called *Typedef* in the OBO Format) from a single, consistently maintained OBO ontology file. ONTO-PERL even provides some extra support

by adding the *is_a* relation type for subsumption in case it is missing in the Typedef section of the OBO file. However, a useful translation of the relation types from the whole set of OBO ontology files to a single RDF store depends on the quality of every single OBO ontology.

The relation types play a key role in the integration of different data sources. Many interesting queries (for instance, "which proteins are *located_in* the cell wall, or any *part_of* it?" or "what regulates the DNA replication?") might exploit the information that is contained in the relations connecting the terms. The OBO Foundry has a policy of using a small set of shared relations for the different ontologies; the implementation of the policy is, however, far from complete. In order to create a self-sustaining ontology file, all the relation types need to be included in the same file, with a name and a unique identifier. This has led to redundant sections for the relations of the 44 imported OBO candidate ontology files, with several inconsistencies. The Relationship Ontology (RO) within the OBO Foundry was designed to create the necessary consistency, but only a fraction of the ontology engineers have considered to use RO. Some relations with the same unique identifiers had different names (*e.g. part_of* and *is_part_of*). This complicates their usage and in particular the process of building queries over different resources that should ideally share the same relation. Moreover, the identifiers cannot serve for the communication with users, as some of them were not meant to be human readable, like BSPO_0000095 (Spatial Ontology [176]) or DESCINHERM (Worm Anatomy Ontology [177]).

6.4.2 Biorel

There is a good reason why many OBO ontology engineers decided not to use RO for their project. RO contains only a limited number of relation types that are very generic and often even cryptic. These relation types are investigated in the Basic Formal Ontology (BFO) project and their definitions are very technical. For instance, the distinction between the relation types *located in*, *contained in* and *is part of*, have generated many discussions, often leading to different conclusions. Moreover, all the identifiers of the relation types that are used within RO include a strange prefix *OBO_REL* that does not fit anywhere in the rest of the OBO Format. In order to avoid any problems with this, most developers of OBO ontologies have chosen to create their own relation types.

Biorel (name deriving from: *Bio*medical *rel*ation) was created as an

OBO formatted relation ontology that contains all the relation types that were defined anywhere within the OBO project (including OBO Foundry ontologies like GO, OBO Foundry candidate ontologies like NCBI Taxonomy and OBO formatted ontologies like CCO). Every relation type appears just once and is given a consistent name and identifier. The naming system is one of the primary functions of Biorel within BioGateway, since all the biomedical relation types in BioGateway could be mapped to relation types in Biorel. All the relation types were given a name that contains a verb conjugated in the third person singular in Biorel. For some reason, most OBO ontology engineers had not created an appropriate name for the relation types and they have just copied the identifier into the name tag, including underscores and cryptic notations. The following section shows how Biorel has dealt with the names:

```
[Typedef]
id: binds_to
name: binds to

[Typedef]
id: bounds
name: bounds

[Typedef]
id: branch_of
name: is branch of

[Typedef]
id: broader
name: is broader than

[Typedef]
id: BSPO:0000096
name: is anterior to
```

Such names are much better suited to be returned as answers to queries as compared to the cryptic identifiers. The application of this naming system predominantly involved the addition of the verb *is*. As a consequence, we can return triples in the form of a *pseudo*-grammatical sentence, like *blood is located in vein*. This rule also prompted the transformation of names like *anatomical_relation* into *is anatomically related to* and *surrounding* into *surrounds*. In fact, the meaning of several poorly named relation types became clearer by adhering to this format. Biorel also contains all the other OBO tags for relation types, with metadata like definitions,

synonyms and comments, as well as the semantic properties, like transitivity, reflexivity, inverses and chains. The following three examples are relation types that have technical definitions resulting from the discussions around BFO:

```
[Typedef]
id: bearer_of
name: is bearer of
def: "A relation between an entity and a dependent
continuant; the reciprocal relation of inheres_in"
[GOC:cjm]
comment: Examples: red eye bearer_of redness
synonym: "has_inherent" EXACT []
synonym: "has_inherer" EXACT []
inverse_of: inheres_in ! inheres in

...

[Typedef]
id: is_a
name: is
builtin: true
def: "For continuants: C is_a C' if and only if: given
any c that instantiates C at a time t, c instantiates C'
at t. For processes: P is_a P' if and only if: that given
any p that instantiates P, then p instantiates P'."
[PMID:15892874]
comment: The is_a relationship is considered axiomatic by
the obo file format specification, and by OWL
synonym: "is_subtype_of" RELATED []
xref: rdfs:subClassOf
is_anti_symmetric: true
is_reflexive: true
is_transitive: true

...

[Typedef]
id: part_of
name: is part of
def: "For continuants: C part_of C' if and only if: given
any c that instantiates C at a time t, there is some c'
such that c' instantiates C' at time t, and c *part_of* c'
at t. For processes: P part_of P' if and only if: given
any p that instantiates P at a time t, there is some p'
such that p' instantiates P' at time t, and p *part_of* p'
at t. (Here *part_of* is the instance-level
```

```
part-relation.)" [PMID:15892874]
is_reflexive: true
is_transitive: true
inverse_of: has_part ! has part
holds_over_chain: results_in_formation_of ends_during
holds_over_chain:
results_in_complete_development_of starts_during
```

The translation of the relation types to RDF for BioGateway was maintained manually through the use of Biorel. The translation of the Typedef sections in the OBO files was simply skipped in the automated translation pipeline, and the RDF export of Biorel (biorel.rdf) was loaded into every graph. Biorel was also loaded in graphs that did not contain OBO files, because the relation types of the OBO Foundry are designed to be used within any source containing biomedical knowledge. The GOA-associations essentially consist of relations between proteins in UniProt and terms in the Gene Ontology (GO). These relations were easily mapped to the OBO relation types *has function*, *is located in*, and *participates in* for the GO molecular functions, GO cellular components and GO biological processes respectively.

The skipping of the Typedef sections is another example of the practice to avoid that the addition of triples to the knowledge base disables basic queries. Before this operation, queries got heavily corrupted due to multiple labels for a single relation type, not to speak of the cryptic labels that were hard to comprehend.

6.4.3 BioMetarel

Creating user friendly names for the relation types is just a tiny part of the relation management. A more interesting feature is the exploitation of the relation type semantics for reasoning. For this reason, Metarel itself was also translated into RDF. This would enable the management of the relation types in Biorel. The graph that integrates Metarel and Biorel was called *BioMetarel*.

BioMetarel contains a maximum of the semantics that is required to execute automated reasoning, by a classification of relations, a classification of relation types and a classification of axioms about relation types. The generic sections of these three classifications come from Metarel, whereas the relation types come from Biorel.

The creation of BioMetarel consists of three steps:

- Loading the RDF exports of Metarel and Biorel in the same RDF graph.
- Loading a manually maintained set of links between Biorel en Metarel, in the form of RDF triples.
- Inferring all the possible directly available conclusions about the semantics of the relation types through automated inference procedures.

The first step is trivial, but the second and the third require some more explanation. The second step, the loading of a manually maintained set of links, expresses the fact that not all the semantics of the relation types in Biorel was properly encoded within the Biorel file. The most prominent missing information about the relation semantics is the absence of any distinction between class relation types and instance relation types. The OBO Foundry works with class relation types, but different semantic rules (for instance inverses and symmetry) are expressed for the corresponding instance relation type. This practice is not a surprise, since most paradigms in Logic, like Predicate Logic, DL and OWL, work with instance relation types (known as roles or properties) exclusively. This has influenced the work of many ontology engineers working on OBO ontologies.

An adequate treatment of the relation semantics in BioMetarel has necessitated the manual creation of two sets of relation types, one for the class relation types and the other for the instance relation types, with links between them. Another reason for creating such a manually maintained set of links is to formalize the many informally expressed relation type axioms that ontology engineers have stated within the 'comment' tag of the Typedef stanzas in their ontologies. At least this was the case for the version of BioMetarel that was used for the first two publications of BioGateway in 2008 and 2009. Later, when Biorel was translated into OWL 2 DL instead of plain RDF, the informal rules were formalized immediately in Biorel, which has reduced the extra utility of BioMetarel mainly to the correct treatment of class relation types. All the links were assembled within a single RDF file, edited in the turtle syntax, called biometarel_merge.rdf.turtle (see Appendix BioMetarel).

The third step, the inferring of directly available conclusions within BioMetarel, consists of the actual classification of relations, relation types and relation type axioms that are used in BioGateway.

The classification of relations has relations as instances and relation types as classes. It corresponds to the role hierarchy in Description Logic. The relation types in this hierarchy are connected by the predicate *rdfs:subPropertyOf*. This is a transitive predicate: for example 'activates' is a subproperty of 'positively regulates' and 'positively regulates' is a subproperty of 'regulates'. This implies that 'activates' is a subproperty of 'regulates'. However, the triple 'obo:activates rdfs:subPropertyOf obo:regulates' is not explicitly present in BioMetarel without the inferring of directly available conclusions in the classification of relations. The following SPARQL/Update (SPARUL) query creates the inferences of the subproperty predicate in the classification of relations:

```
sparql
BASE   <http://www.semantic-systems-biology.org/>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl:<http://www.w3.org/2002/07/owl#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX ssb:<http://www.semantic-systems-biology.org/SSB#>
PREFIX obo:<http://purl.obolibrary.org/obo/>
PREFIX metarel:<http://www.metarel.org/>
INSERT INTO GRAPH <biometarel> {
  ?node1 rdfs:subPropertyOf ?node3.
}
WHERE {
  GRAPH <biometarel> {
    ?node1 rdfs:subPropertyOf ?node2.
    ?node2 rdfs:subPropertyOf ?node3.
  }
}
```

This update query has to be iterated recursively until there are no new triples that can be inferred.

Also the classification of relation types requires extra inference. For example all the relation types that are transitive should be classified as an instance of Metarel's metaclass *transitive relation type*. We know that the inverse of a transitive relation type is also transitive. This information is not always explicitly present in Biorel. For instance, the relation type *precedes* is annotated as transitive, but the relation type *is preceded by* is not. The following SPARUL update query will create this inference:

```
sparql
# BioMetarel creation: Create a transitive relation
# type when the inverse is transitive
```

```
BASE   <http://www.semantic-systems-biology.org/>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl:<http://www.w3.org/2002/07/owl#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX ssb:<http://www.semantic-systems-biology.org/SSB#>
PREFIX obo:<http://purl.obolibrary.org/obo/>
PREFIX metarel:<http://www.metarel.org/>
INSERT INTO GRAPH <biometarel> {
  ?transitive_rt rdf:type metarel:TransitiveRelationType.
}
WHERE {
  GRAPH <biometarel> {
    ?transitive_rt owl:inverseOf ?transitive_inverse.
    ?transitive_inverse rdf:type owl:TransitiveProperty.
  }
}
```

When an instance relation type is transitive, then also a class relation type that is based on the instance relation type is transitive. This rule propagates transitivity to the level of class relation types within the relation type classification:

```
sparql
# BioMetarel creation: Create the transitive relation
# types as children of transitive_relation_type
BASE   <http://www.semantic-systems-biology.org/>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl:<http://www.w3.org/2002/07/owl#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX ssb:<http://www.semantic-systems-biology.org/SSB#>
PREFIX obo:<http://purl.obolibrary.org/obo/>
PREFIX metarel:<http://www.metarel.org/>
INSERT INTO GRAPH <biometarel> {
  ?transitive_rt rdf:type metarel:TransitiveRelationType.
}
WHERE {
  GRAPH <biometarel> {
    {?transitive_rt rdf:type owl:TransitiveProperty}
    UNION {
    ?transitive_rt metarel:isBasedOn ?transitive_irt1.
    ?transitive_irt1 rdf:type owl:TransitiveProperty.}
  }
}
```

An example of inference within the classification of relation type axioms is the propagation of chain rules from the level of instance relation

types to the level of class relation types, and the correct annotation of the chain rules in Metarel. The chain rules are extracted from the rather complicated OWL/RDF model in biorel.owl and added to the Metarel model in the BioMetarel graph:

```
sparql
# Propagate chains between instance relation types to the
# level of class relation types.
BASE   <http://www.semantic-systems-biology.org/>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl:<http://www.w3.org/2002/07/owl#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX ssb:<http://www.semantic-systems-biology.org/SSB#>
PREFIX obo:<http://purl.obolibrary.org/obo/>
PREFIX metarel:<http://www.metarel.org/>
INSERT INTO GRAPH <biometarel> {
  _:construct metarel:hasFirstRelation   ?First_RelationType.
  _:construct metarel:hasSecondRelation ?Second_RelationType.
  _:construct metarel:hasResultingRelation
                                      ?Resulting_RelationType.
}
WHERE {
 GRAPH <biometarel> {
  ?Resulting_InstanceRelationType owl:propertyChainAxiom
                                                      ?node.
  ?node rdf:first ?First_InstanceRelationType.
  ?node rdf:rest ?rest.
  ?rest rdf:first ?Second_InstanceRelationType.
  ?First_RelationType     metarel:isBasedOn
                             ?First_InstanceRelationType.
  ?Second_RelationType    metarel:isBasedOn
                           ?Second_InstanceRelationType.
  ?Resulting_RelationType metarel:isBasedOn
                            ?Resulting_InstanceRelationType.
  ?First_RelationType     rdf:type
                            metarel:AllSomeClassRelationType.
  ?Second_RelationType    rdf:type
                            metarel:AllSomeClassRelationType.
  ?Resulting_RelationType rdf:type
                            metarel:AllSomeClassRelationType.
 }
}
```

The creation of BioMetarel in this manner is only a preparation for the reasoning process. These three steps, the loading, the merging and the classifications, affect only the set of about 7500 triples in BioMetarel. The

reasoning process itself may affect all the hundreds of millions of triples in the RDF store. Meanwhile, even without the execution of any reasoning rules, BioMetarel can also serve to answer questions about the semantics of the relation types. This is a useful asset for the bioinformatics oriented users of BioGateway, since the semantics of relation types is heavily debated among ontologists, logicians and even biologist users of the OBO ontologies.

6.4.4 Transitive closures

The full-scale reasoning approach was not yet established for the first publication of BioGateway in 2008. There was, however, already an important step into this direction through the automated inferring of triples that are implied by the transitivity of relations. The complete inference of triples by this procedure, up to the point that not any extra triple can be inferred, is called a *transitive closure*.

Three methods for creating the transitive closures were taken into consideration. They can be created with:

- a single SPARUL update query on an engine that uses recursive inferences
- a sufficiently large loop on this SPARUL update query, on an engine that does not use recursion
- ONTO-PERL, a programming interface in the PERL language

The first method was used for BioGateway on an engine with a version of SPARUL that supports the recursion. The second method was also tested successfully for BioGateway. It takes, however, more time to do the reasoning in this way, and it may fail to work on ontologies with transitive chains that exceed the predefined length. The SPARUL update query for creating the closure for the first two methods is the following:

```
sparql
BASE   <http://www.semantic-systems-biology.org/>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl:<http://www.w3.org/2002/07/owl#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX ssb:<http://www.semantic-systems-biology.org/SSB#>
PREFIX obo:<http://purl.obolibrary.org/obo/>
PREFIX metarel:<http://www.metarel.org/>
```

```
INSERT INTO GRAPH <gene_ontology_edit_tc> {
  ?class1 ?rel_id ?class3.
}
WHERE {
  GRAPH <gene_ontology_edit_tc> {
    ?class1 ?rel_id ?class2.
    ?class2 ?rel_id ?class3.
  }
  GRAPH <biometarel> {
    ?rel_id rdf:type metarel:TransitiveRelationType.
  }
}
```

The update query searches within the Gene Ontology for all the cases in which a class1 is connected to a class2 through a transitive relation type and in which this class2 is connected to a class3 through the same transitive relation type. It will create a new relation arc, declaring this relation type, between class1 and class3 in the RDF graph of the Gene Ontology that has the suffix _tc.

In the case of the term *leptotene* (GO:0000237), which is originally only linked to the terms cell cycle phase (GO:0022403) via an *is_a* relation and *meiotic prophase I* (GO:0007128) via a *part_of* relation, the following implicit relations are added: *leptotene is_a biological_process* (GO:0008150), *leptotene is_a cellular process* (GO:0009987), *leptotene is_a cell cycle process* (GO:0022402), *leptotene is_a cell cycle phase* (GO:0022403), *leptotene part_of meiosis* (GO:0007126), *leptotene part_of meiosis I* (GO:0007127), *leptotene part_of meiotic prophase I* (GO:0007128), *leptotene part_of meiotic cell cycle* (GO:0051321) and *leptotene part_of M phase of meiotic cell cycle* (GO:0051327) (see Figure 6.3).

There are two possible interpretations of an update query. Either the update query is applied just once on the original data (atomic updates), or the update query is executed again after the addition of new triples that were inferred by the update query. The second interpretation uses recursion and is slightly more advanced than the first. It is a form of *monotonic* reasoning, as long as no triples are ever deleted and as long as no conclusions are ever derived from the absence of any triples in the store.

The latest releases of the Virtuoso software [153], from Virtuoso 6 and onwards, use only atomic updates and no recursion. This is in accordance with the W3C specification for SPARUL, which was never designed to execute fully automated reasoning. Earlier versions of Virtuoso, at least the release of Virtuoso 5.0.8, use the recursive interpretation for certain

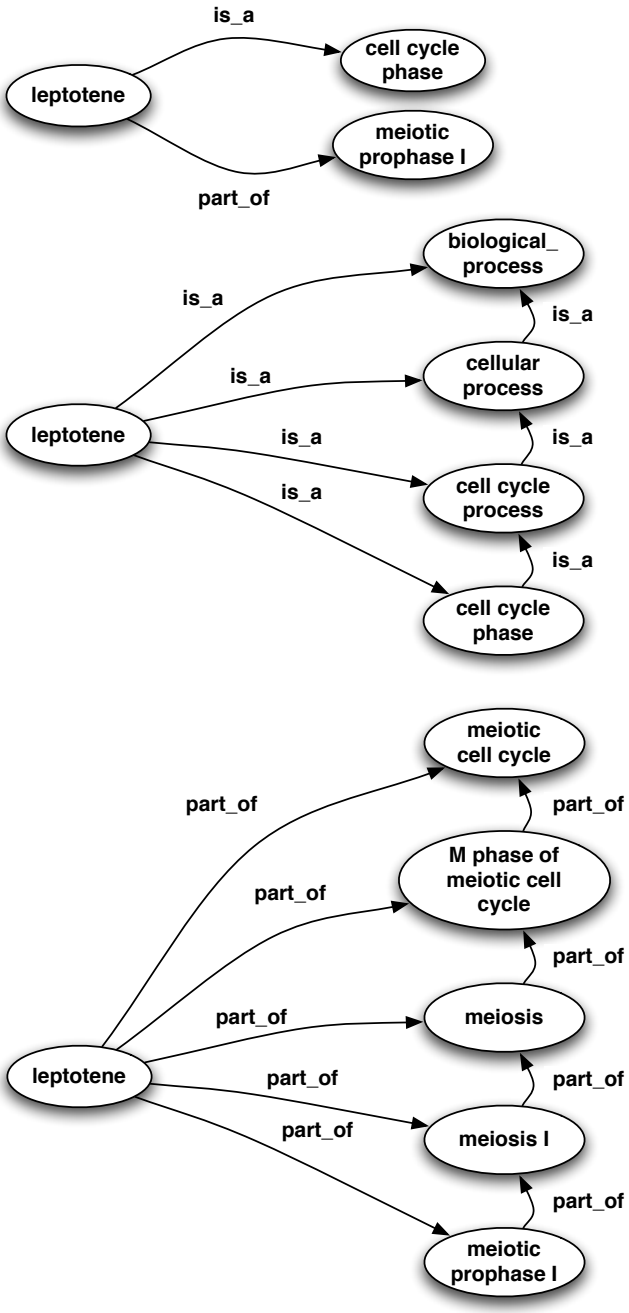


Figure 6.3: An example of a transitive closure for the term *leptotene*. Many new triples are inferred from the two original triples in the top figure.

types of update queries. This has been very practical to execute the transitive closure without requiring semi-automated approaches like using a loop that runs the same update query many times.

A method for creating transitive closures was also developed by Erick Antezana in ONTO-PERL [1]. This subroutine still exists as a solid building block for more advanced and flexible reasoning through procedural methods in the Perl programming language. ONTO-PERL creates the inferences from transitivity and reflexivity in the RDF document before its upload to the triple store. Because of this, the reasoning abilities are not limited to the RDF tools that are included in the software layers of the RDF store. However, the practice turned out to be much slower. Another problem was the fact that the semantics of the relation types, transitivity in this case, were hard-coded inside the subroutines of ONTO-PERL, instead of managing them in an RDF file like in the case of BioMetarel. This has put a burden on the continuation of this approach to reasoning beyond the transitivity of the two most important relation types: *is_a* and *part_of*. The usage of BioMetarel within ONTO-PERL has been identified as a possible solution to overcome this problem.

6.5 MetaOnto

RDF is best known as a representation format that consists of triples and an RDF store is consequently often called a *triple store*. However, strictly speaking, RDF consists of *quads* instead of triples and many developers who are acquainted with the lower levels of the architecture of RDF store speak of *quad stores*. The fourth identifier that is required in addition to the first three (the subject, the predicate and the object) is the identifier of the graph to which a triple belongs. Whereas a triple consisting of a subject, a predicate and an object is always unique within an RDF graph, a quad, consisting of a subject, a predicate, an object and a graph, is unique within an RDF store.

The graphs in the quads can be queried with SPARQL and SPARUL through a slightly different syntactical expression. All the subjects, predicates and objects in quads that belong to the same graph, are fitted between curly brackets that are preceded by 'GRAPH' and the graph IRI or graph variable. This results in the next SPARQL query to return all the quads in the RDF store:

```
SELECT ?s ?p ?o ?g
WHERE {
 GRAPH ?g {
  ?s ?p ?o.
 }
}
```

Since BioGateway exploits the usage of different graphs, this kind of queries can be quite important. There is a separate graph for every of the nearly 2000 species annotated in the GOA project, as well as for about 80 ontologies in the OBO Foundry. In order to enable certain SPARQL queries, some information about all these graphs in BioGateway was assembled in a separate RDF graph: MetaOnto.

For the OBO ontologies, a little OBO formatted meta-ontology was created with information about each OBO ontology. The first entries are shown here:

```
format-version: 1.2
date: 30:07:2008 15:14
saved-by: wablo
auto-generated-by: OBO-Edit 1.002
remark: MetaOnto is a meta-ontology that has OBO ontologies
as terms.

[Term]
id: METAONTO:0000001
name: OBO ontology
def: "An OBO ontology is a science-based ontology that was
established following a set of principles for ontology
development with the goal of creating a suite of orthogonal
interoperable reference ontologies in the biomedical
domain." [http://www.obofoundry.org/]

[Term]
id: METAONTO:0010001
name: Amphibian gross anatomy
def: "A structured controlled vocabulary of the anatomy of
Amphibians." [http://www.obofoundry.org/cgi-bin/
detail.cgi?id=amphibian_anatomy]
synonym: "AAO" EXACT []
is_a: METAONTO:0000001 ! OBO ontology

[Term]
id: METAONTO:0010002
name: Spatial Ontology
def: "A small ontology for anatomical spatial references,
```

```
such as dorsal, ventral, axis, and so forth."
[http://www.obofoundry.org/cgi-bin/detail.cgi?id=spatial]
synonym: "BSPO" EXACT []
is_a: METAONTO:0000001 ! OBO ontology

[Term]
id: METAONTO:0010003
name: Common Anatomy Reference Ontology
def: "The Common Anatomy Reference Ontology (CARO) is being
developed to facilitate interoperability between existing
anatomy ontologies for different species, and will provide
a template for building new anatomy ontologies. CARO will
be described in Anatomy Ontologies for Bioinformatics:
Principles and Practice Albert Burger, Duncan Davidson and
Richard Baldock (Editors)"
[http://www.obofoundry.org/cgi-bin/detail.cgi?id=caro]
synonym: "CARO" EXACT []
is_a: METAONTO:0000001 ! OBO ontology
```

The OBO formatted meta-ontology was included in the RDF translation pipeline and uploaded in BioGateway. More information was added subsequently with SPARUL in the following way:

```
BASE   <http://www.semantic-systems-biology.org/>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl:<http://www.w3.org/2002/07/owl#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX ssb:<http://www.semantic-systems-biology.org/SSB#>
PREFIX obo:<http://purl.obolibrary.org/obo/>
PREFIX metarel:<http://www.metarel.org/>
INSERT INTO GRAPH <metaonto> {
  ?graph ssb:instance_of ssb:graph.
}
WHERE {
  GRAPH ?graph {
    ?s ?p ?o.
  }
}
```

This query puts the graph IRI of every BioGateway graph into MetaOnto, as an instance of the meta-class *ssb:graph*. It is followed by the next SPARUL update query:

```
BASE   <http://www.semantic-systems-biology.org/>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl:<http://www.w3.org/2002/07/owl#>
```

```
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX ssb:<http://www.semantic-systems-biology.org/SSB#>
PREFIX obo:<http://purl.obolibrary.org/obo/>
PREFIX metarel:<http://www.metarel.org/>
INSERT INTO GRAPH <metaonto> {
  ?graph ssb:about_taxon ?taxon.
}
WHERE {
  GRAPH ?graph {
    ?term_id ssb:has_source ?taxon.
  }
}
```

This update query adds information about organism taxa. It crawls through all the graphs to look which taxa are used and adds this information to the MetaOnto graph. Naturally, the crawling action should happen after all the graphs are uploaded. The update query was run after the uploading of the original graphs and before the creation of the tc-graphs with inferred closures. The following SPARQL query is enabled by the contents of MetaOnto:

```
# NAME     : Get list of organisms
# FUNCTION : returns all the annotated organisms in the
#          : knowledge base

BASE   <http://www.semantic-systems-biology.org/>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX ssb:<http://www.semantic-systems-biology.org/SSB#>
SELECT distinct ?taxon ?graph
WHERE {
  GRAPH <metaonto> {
    ?graph ssb:about_taxon ?taxon_id.
  }
  GRAPH <ncbi> {
    ?taxon_id rdfs:label ?taxon.
  }
  FILTER(?graph != <SSB>).
}
ORDER BY ?taxon
```

The query returns all the organism taxa in BioGateway and the graphs in which these taxa are used, except for the big SSB graph that contains all the contents. Some results are shown in the following table:

taxon	graph
'Nostoc azollae' 0708	http://www.semantic-systems-biology.org/32899.N_azollae
Abiotrophia defectiva ATCC 49176	http://www.semantic-systems-biology.org/33964.A_defectiva
Acaryochloris marina MBIC11017	http://www.semantic-systems-biology.org/30122.A_marina
Acetobacter pasteurianus IFO 3283-01	http://www.semantic-systems-biology.org/34577.A_pasteurianus
Acetobacter pasteurianus IFO 3283-03	http://www.semantic-systems-biology.org/34579.A_pasteurianus_IFO_3283-03
Acetobacter pasteurianus IFO 3283-07	http://www.semantic-systems-biology.org/34581.A_pasteurianus_IFO_3283-07
.	.
.	.
.	.
Holdemania filiformis DSM 12042	http://www.semantic-systems-biology.org/33910.H_filiformis
Homo sapiens	http://www.semantic-systems-biology.org/25.H_sapiens
Homo sapiens	http://www.semantic-systems-biology.org/9.C_elegans
Hydrogenivirga sp. 128-5-R1-1	http://www.semantic-systems-biology.org/32152.H_sp
Hydrogenobaculum sp. Y04AAS1	http://www.semantic-systems-biology.org/31481.Hydrogenobaculum_sp
.	.
.	.
.	.
Yersinia pseudotuberculosis PB1Yersinia rohdei ATCC 43380	http://www.semantic-systems-biology.org/34113.Y_rohdei
Yersinia ruckeri ATCC 29473	http://www.semantic-systems-biology.org/34822.Y_ruckeri
Zymomonas mobilis	http://www.semantic-systems-biology.org/20760.Z_mobilis_CP4
Zymomonas mobilis subsp. mobilis ATCC 10988	http://www.semantic-systems-biology.org/34234.Z_mobilis_ATCC_10988
Zymomonas mobilis subsp. mobilis NCIMB 11163	http://www.semantic-systems-biology.org/34729.Z_mobilis_NCIB_11163

This list may help researchers to run other queries in the library of BioGateway on more specific graphs that relate to the organism of their

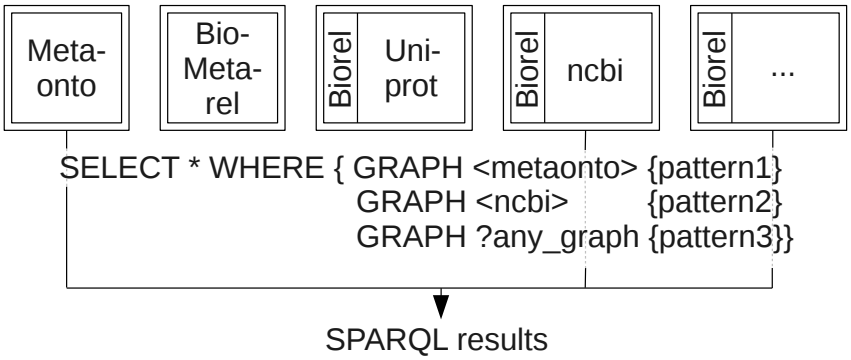


Figure 6.4: An example that shows the generic pattern for querying BioGateway through SPARQL.

interest. Most organism taxa appear only in a single graph, but apparently the graph http://www.semantic-systems-biology.org/9.C_elegans for *Caenorhabditis elegans* also contains the human protein NLRP5 with exactly the same annotations as in the graph for *Homo sapiens*. This is caused by an irregular annotation in this GOA file.

6.6 The library of queries

The library of queries that was developed for BioGateway consists of two sections: biomedical queries and ontological queries. They are represented in that order in the drop-down box on the querying page, where users can investigate the knowledge in BioGateway through SPARQL queries. Retrieving biomedical knowledge is the primary goal for the users of BioGateway, however, other knowledge engineers and bioinformatics researchers may find it useful to find some technical queries that relate to Semantic Web ontologies in general. These technical queries can be found in the section with the ontological queries.

The general pattern for querying can be derived from an example query in Figure 6.4, which integrates the metadata in MetaOnto with taxonomic knowledge from NCBI and with detailed knowledge from any of the specific graphs in BioGateway.

6.6.1 Ontological queries for knowledge engineers

The ontological queries are listed after the biological queries on the querying page, however, they were developed first. They are more basic and are presented first here. Every query was given a number preceded by 'Ont' for the ontological queries, as well as a description of the query. Here follows the list of all the descriptions of the ontological queries:

Ont 1. Query the OBO Foundry: search on names and get their unique id's.
Ont 2. Get all the neighbor terms of a given term.
Ont 3. Get all the properties, like definition, synonyms, etc., of a given OBO term.
Ont 4. Get the names of the graphs in BioGateway.
Ont 5. Get a list of all the ontologies in the OBO Foundry.
Ont 6. Get the hierarchy to the root for a given term.
Ont 7. Get a list of all the relation types that are used in the OBO Foundry.
Ont 8. Get the root term(s) of an ontology.
Ont 9. Get all the children of a given term.
Ont 10. Get all the parents of a given term.
Ont 11. Find all the terms from two different ontologies with the same names.
Ont 12. Get a list of terms in the OBO Foundry that do not have a definition.
Ont 13. Count the amount of triples in a graph.
Ont 14. Count the amount of terms in an ontology.
Ont 15. Search trough all the content on one or more strings.
Ont 16. Get a limited list of terms from a graph.
Ont 17. Get all the terms that are 'part of' a given term.
Ont 18. Get the hierarchy above a certain term for a given transitive relation type.
Ont 19. Get information about relation types.
Ont 20. Get the closest common parent in the hierarchy.
Ont 21. Compare direct with inferred annotations.
Ont 22. Count the number of triples in BioGateway.

This library is effectively providing an entrance to all the data in BioGateway, and to the possible ways in which it can be queried. Without the ontological queries, a knowledge engineer would have to explore BioGateway by trial and error. Now she might rearrange pieces of the code in the

query to build new queries. Ont 1, the first query, shows which syntax to use for making a text-based search in the OBO-ontologies:

```
# NAME      : Search terms on name
# PARAMETER: blood (see line 8 in the SPARQL code): the name
# FUNCTION : returns all the labeled terms for which the
#            name contains 'blood'

BASE   <http://www.semantic-systems-biology.org/>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX ssb:<http://www.semantic-systems-biology.org/SSB#>
SELECT ?name ?unique_id
WHERE {
  GRAPH <OBO> {
    ?unique_id rdfs:label ?name.
    FILTER regex(str(?name), 'blood')
  }
}
```

6.6.2 Biomedical queries for knowledge exploration

The goal of BioGateway is to enable knowledge exploration for expert life science reseachers. Therefore, the biomedical queries are more important. It is not an easy task to submit useful queries on a large knowledge base. It requires knowledge about the query language (SPARQL), but also about the contents of the resources in the knowledge base. Bridging the gap between these two domains of expertise is the typical task for a Bioinformatics researcher. With the ontological queries in place, the investigation of the biomedical content could start. The construction of the biomedical queries has required discussions with plant systems biologists of VIB in Ghent and scientists working on human signal transduction networks in the NTNU institute of Trondheim.

The following library of biomedical queries was retained for BioGateway:

Bio 0. Get basic information by the unique ID.
Bio 1. Get protein: get the unique ID of a protein by a protein or gene name.
Bio 2. Get functional, locational, process and literature information from GOA about a given protein.
Bio 3. Get the proteins that are involved in a given disease (e.g. psoriasis).
Bio 4. Get the proteins that participate in the same process as a given pro-

tein.
Bio 5. Get the proteins that are located in both the nucleus and the endoplasmic reticulum.
Bio 6. Get the number of interactors for the proteins in a PPI network.
Bio 10. Get all the proteins that are involved in two specific diseases.
Bio 11. Get the proteins that are involved in many diseases.
Bio 12. Get the proteins with a specific function, location and process for all the annotated organisms.
Bio 13. Get the proteins that are involved in a given topic (e.g. insulin).
Bio 14. Get the proteins that are involved in a given topic and a given disease.
Bio 15. Get protein interactors by a given topic and a given disease.
Bio 16. Get the similarities between two proteins.
Bio 17. Get a list of all the organisms in the knowledge base.
Bio 18. Get proteins in protein complexes.
Bio 19. Find the smallest protein complexes.
Bio 20. Get the the proteins that interact in the same protein complex.
Bio 21. Get extracted information from a given publication.
Bio 22. Get the proteins in the nucleus that are involved in diabetes.
Bio 23. Get specifically annotated types of proteins in mammals.

Most of these queries exploit the integration of the Gene Ontology (GO), the Gene Ontology Annotations (GOA), UniProt/Swiss-prot, NCBI taxonomy and PubMed annotations. In spite of the fact that many other OBO ontologies, apart from GO, are present in BioGateway, there are not any example queries in the library that elicit their contents explicitly. Most of the OBO ontologies are not so well integrated with other resources compared to GO, and the investigation of all the different scientific subdomains they represent would not be a fruitful effort. Therefore the queries were oriented towards genome researchers. Many of them stem from the attempt to generate hypotheses about the biological meaning of specific protein interactions.

The first two queries, Bio0 and Bio1, are essential for researchers who are interested in a specific protein type. Bio0 provides extra information through the unique identifier of the protein type within BioGateway, whereas Bio1 can be used to find out the unique identifier through names that are better known. It is clear that names that are known by scientists will never be unique within a system that consists of about 2000 organisms,

that all have thousands of protein types.

The use of the library with biomedical queries can be demonstrated by a series of consecutive queries that a biologist might have. Imagine a researcher who wants to investigate the diabetes disease. She looks through the preconstructed queries in the querying page and finds

- Get the proteins involved in a given disease (e.g. psoriasis).

She clicks this query and changes *psoriasis* into *diabetes* in the edit box. After running this query she gets already a lot of information about diabetes in the disease descriptions that appear. She also finds many proteins that are involved in diabetes, so she can scout for proteins that she knows from her previous experience. Some other terms like *autoantigen* and *non-insulin-dependent diabetes mellitus* attract her attention, so she can use these terms to restrict the query and to take a closer look at these results. As the researcher wants to find out more about the signaling pathways that relate to diabetes, she chooses another preconstructed query:

- Get the proteins involved in a given topic and a given disease.

She tries first with the keywords *pathway* and *diabetes*, for the topic and the disease respectively. She gets three proteins as the result and a lot of information about what kind of pathways they are involved in, and how aberrations of these proteins can cause diabetes. She would also like to see with which other proteins these proteins interact, so she chooses a third query from the drop-down box:

- Get protein interactors by a given topic and a given disease.

She chooses the same keywords, *pathway* and *diabetes*, to find out that the protein INSR_HUMAN has two interactors: SRBS1_HUMAN and PTN1_HUMAN. She chooses the query to get their unique ids to investigate these proteins further:

- Get the unique ID of a protein by its name.

The ids of the proteins are retrieved after copy-pasting the names in the correct place in the query, as indicated in the PARAMETER line. For the PTN1_HUMAN protein, she copies P18031, the UniProt ID, from the result. Now she can use this query:

- Get functional, locational, process and disease information about a given protein.

by copy-pasting the unique id as parameter. The results show that this protein has phosphorylation as function and is also involved in the insulin receptor signaling pathway.

6.7 Visualization of queries

A SPARQL browser enables querying and visual exploration of the result. It can be accessed from the SSB website[6]. With this interface, users can define a SPARQL query over BioGateway resources, the SPARQL endpoint could also be customized. After executing a query, a network of results is displayed (see Figure 6.5). A tabular representation of the result is also available.

The SPARQL browser was developed using Flex technologies [178], which provide powerful ways of creating interfaces with dynamic features. The entire source code is freely available [179].

In spite of being attractive for users, the graphical interface of the SPARQL browser does not make the querying process easier. Writing SPARQL code is still required for any query that goes beyond showing the local neighborhood. Another attempt to make querying in BioGateway easier was done by Brad Chapman, who wrote a Python API interface that queries the SPARQL endpoints of both BioGateway and InterMine [180]. The interface is specifically written on the library of biological queries in BioGateway. However, a generic solution for practical and expressive querying of RDF knowledge bases is something that deserves to be investigated more in the future.

[6] http://www.semantic-systems-biology.org/sparql-viewer

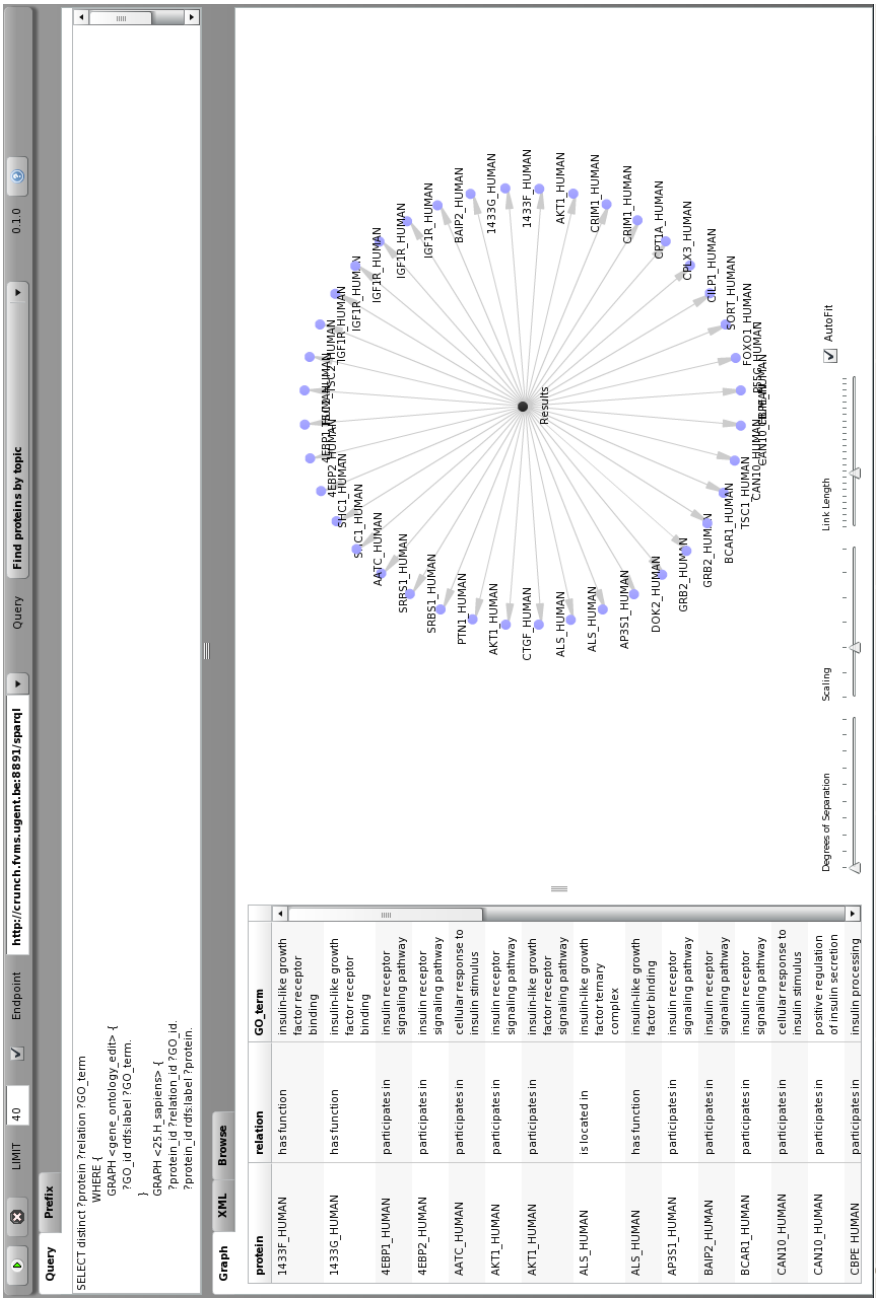


Figure 6.5: Network of SPARQL query results over BioGateway. The query shows proteins that are related to 'insulin', either by their function, by the biological process in which they participate, or by the protein complex they are located in.

Chapter 7

Computational reasoning with Metarel

7.1 Description Logics in three steps

Description Logics research has kept the hope alive for realizing fully automated reasoning on bio-ontologies [181]. This research promises that any ontology that is modeled in a DL language makes unambiguous sense and that an automated reasoner can answer logical questions about the ontology correctly. However, applying the fully fledged reasoning approach on large, integrated bio-ontologies has proven to be overambitious for two main reasons. First of all, the developers of bio-ontologies, often more experienced in biology than computer science, do not succeed to model all the available knowledge into the rigid language constructs of logics. Consequently, bio-ontologies are full of glitches concerning their logic-based rules [182]. Secondly, even if a large bio-ontology succeeds to pass the computational proof of consistency, current automated reasoners are not fast enough for answering queries. Although computer performance continues to increase, the amount of knowledge and data in bioinformatics has been growing even faster.

Even an ontology with imperfections can be useful by providing sensible answers to many real life questions. In order to better exploit the available ontologies, we need an approach that benefits from DL as much as possible, without insisting on the exclusive use of DL at all stages in constructing a practical query system.

Here, the enabling of large-scale reasoning in the knowledge base

BioGateway is approached in three steps: 1) define a logic-based representation language; 2) build a consistent ontology; and 3) create inferences for enabling queries. DL reasoning is very useful in the first two steps and has proven already useful for consistency checking of smaller units. However, it is still problematic to implement DL in a query system on a very large scale.

The third step for the ontologies in BioGateway is accomplished by capitalizing on the prior work (by others and ourselves) with respect to steps one and two. We minimally adjusted this prior work by a manual curation effort that was restricted to the relation types (types of relations like *is part of*, *is located in*) that were used. This curation effort implies a certain feedback from the last step to the previous steps. Certain language constructs, like defined classes, domains and ranges or restrictions on the number of relations of certain type from a given subject, may turn out to be very expensive in terms of query time. Alternatively, a relation type used in a given ontology may turn out to create masses of useless inferences. By using Metarel as a semantic framework, and SPARQL/Update as inference tool, we had ample flexibility to engage in a trial and error process to create only those inferences that were useful and necessary. This is a practical alternative to the ambitious approach of DL to execute reasoning as a one-step process without any flexibility for optimization or feedback.

7.2 Reasoning on bio-ontologies

7.2.1 All-some relations between classes

Biological knowledge consists almost always of relations between classes (groups) of different individual biological entities. When we express knowledge about cells, proteins or organisms, for instance, we are not referring to a single cell that we observe under a microscope, or a particular mouse that was injected yesterday. We rather refer to classes of many entities that behave in similar ways, and these classes are what we name and identify. In comparison, for example the geographical knowledge domain is strikingly different, as the Atlantic Ocean, New York and Bermuda are large and significant enough to be referred to as individuals with a (usually capitalized) proper name and a proper identifier.

In queries about biological knowledge, we need a logical semantics for relations between classes. The all-some interpretation is the most

prominent example to illustrate this [38]. When we relate the classes 'p53-protein' and 'tumor suppression' with the *has function* relation, it has to mean that all p53-proteins have some tumor suppression as function. This way of using relations provides a very powerful system to infer sound statements of biological knowledge.

Let us look at an example using two statements that have the all-some interpretation: 'every p53-protein *is* some protein' and 'every protein *is encoded by* some gene'. From these two we can derive logically that 'every p53-protein *is encoded by* some gene'. The inferred statement is sound and it may be the basis for further conclusions.

All the millions of biological classes and the relations between them can be represented as a large network or graph. Queries can be constructed by defining a pattern or subgraph that must match one or several segments of the larger network. Imagine we want to find all the objects that are encoded by a gene and that have some tumor suppression function. Then the search pattern will consist of two triples and one subject that we are interested in: 'my subject *is encoded by* gene' and 'my subject *has function* tumor suppression'. This pattern should match sections of the network, with the middle part of the triples fitting to the relations and the binding elements to the biological classes. The subject 'p53-protein' is a possible answer to the query.

This example is useful to demonstrate the importance of reasoning. What happens if nobody bothered to add 'every p53-protein *is encoded by* some gene' explicitly? This absence would prohibit finding 'p53-protein' among the list of answers, although this statement follows logically from the two statements described above. It appears that in a good knowledge system all sound statements that are implicit should be made explicit by logical inference, thus augmenting the explicit knowledge in the system by pre-computing. A complete inference of implicit knowledge can be referred to as a 'closure'.

7.2.2 Five closure rules for inferring all-some relations

Five closure rules for inferring knowledge statements concerning relations between biological classes with an all-some interpretation are proposed here. These five rules together provide the foundation for the reasoning in step 3 on the current state-of-the-art OBO ontologies and on annotations with OBO ontologies. Annotations of biological subjects imply that an ontology relation and an ontology term are used in the second and third parts

of a knowledge statement that is represented as a triple.

Let A, B and C be classes and R, S and T be relation types. For instance, with A = 'p53-protein', R = '*is encoded by*' and B = 'gene', the knowledge statement $A\ R\ B$ means 'Every p53-protein *is encoded by* some gene'.

1. **Reflexivity**
   A reflexive closure infers the knowledge statements $A\ R\ A$, where R is a reflexive relation type. For instance, 'every body *is part of* some body'.

2. **Transitivity**
   A transitive closure infers the knowledge statements $A\ R\ C$, when the knowledge statements $A\ R\ B$ and $B\ R\ C$ exist and R is a transitive relation type. For instance, 'every kidney *is located in* some body' follows from 'every kidney *is located in* some abdomen' and 'every abdomen *is located in* some body'.

3. **Priority over subsumption**
   The priority over subsumption infers the knowledge statement $A\ R\ C$, if A is a subclass of B and the knowledge statement $B\ R\ C$ exists, or if the knowledge statement $A\ R\ B$ exists and B is a subclass of C. For instance, 'every API5-protein *regulates* some cell death' follows from 'every API5-protein *regulates* some apoptosis' and 'every apoptosis *is* some cell death'.

4. **Super-relations**
   A knowledge statement $A\ S\ B$ is inferred if S is a super-relation of R and the knowledge statement $A\ R\ B$ exists. For instance, 'every API5-protein *regulates* some apoptosis' follows from 'every API5-protein *negatively regulates* some apoptosis'.

5. **Chains**
   A knowledge statement $A\ R\ C$ is inferred if the knowledge statements $A\ S\ B$ and $B\ T\ C$ exist and R holds over a chain of S and T. The relation types R, S and T do not need to be all different. For instance, 'every API5-protein *negatively regulates* some apoptosis' follows from 'every API5-protein *participates in* some anti-apoptosis' and 'every anti-apoptosis *negatively regulates* some apoptosis'.

These five closure rules allow to infer most of the implicit knowledge contained in the Gene Ontology (GO), the NCBI Taxonomy, OBO ontologies that were built similarly to GO, as well as in annotations made with such ontologies (like GOA), and make it available to querying. They could be expressed as four rules if transitivity were modeled as a chain where the three relation types R, S and T are all identical.

Such closure rules for all-some relations between classes follow directly from rules expressed for (chains of) relations between instances, which are common for Description Logics and OWL. Indeed, if all instances from class A are related to some instances of class B and all instances of class B are related to some instances of class C, then all instances of class A are connected by a chain of two instance relations to some instances of class C. The language features corresponding to the closure rules within step 3 (DL: chains as role constructors; global reflexivity for atomic roles, transitivity for atomic roles and role inclusions as role axioms; existential restrictions of atomic concepts by a role as concept constructors; and concept inclusions with atomic concepts on the left-hand side as terminological axioms) are a subset of those in OWL 2 EL and OWL 2 DL, which warrants efficient reasoning in a decidable semantics.

The semantics of OBO implies additional rules for inferring new knowledge statements. A prominent asset is the use of classes that are logically defined from primitive classes (DL: atomic concepts) through necessary and sufficient conditions. Such defined classes are used in most DL languages, like OWL DL, OWL EL and OWL RL. OBO ontologies have mostly primitive classes with natural language definitions, although logical definitions through intersections of classes are also used. However, the rules in step 3 treat an all-some relation between classes only as a necessary condition, which is not enough for a logical definition.

Other features in OBO that do not appear in step 3 are domains, ranges, symmetry, union, disjointness and functionality of relation types, and union and disjointness of classes. It is inherent to the idea introduced above (separating reasoning in three steps) that rules for these features were applied already in step 2 (building a consistent ontology).

Step 3 uses language features that can express knowledge more compactly (DL: logic entailment) and avoids the reasoning problems associated with consistency checking for logically defined classes (DL: satisfiability). If for instance the relation type *precedes* was given the range *process* and some annotator or ontology engineer erroneously creates a *precedes* rela-

tion to a logically defined class that is disjoint from *process*, then a reasoner should detect this problem in step 2.

An issue not mentioned here is the treatment of individuals (DL: assertional axioms), because they are currently not used in OBO ontologies, nor in the biomedical KBs that are annotated with OBO ontology classes. The individual geographical entities present in the OBO ontologies are modeled as singleton classes. In order to model and treat them as individuals, the five rules would need to be complemented with some extra rules. Inverses of relation types, which do not have logical consequences on all-some relations between classes, might also be of use in this extension. For example, from 'Ghent is part of Europe', we can infer 'Europe has part Ghent'. A similar conclusion cannot be derived for classes, like in 'uterus is part of abdomen'. A logical reasoner cannot derive whether every abdomen has a uterus as part or not.

7.3 Methods

The large-scale inference of biological knowledge statements was achieved with RDF tools, operating on a merger of Metarel and BioGateway. The different steps in the entire reasoning pipeline are visualized in Figure 7.1.

The merging was relatively straightforward, as the ontologies in BioGateway consisted of the simple triple form subject-relation-object. We curated all the relation types that were used in the OBO ontologies, both candidates and adopted ones, assembling them in a relation ontology called *biorel.obo*. Subsequently, we translated *biorel.obo* into OWL 2 DL and merged it as an RDF graph with *metarel.rdf*. This resulted in the relation graph *biometarel.rdf* for use in BioGateway. Finally, we inferred new knowledge statements as RDF triples by running SPARQL/Update queries over both *biometarel.rdf* and the existing RDF graphs in BioGateway, thereby executing the above-described reasoning approach. The most important files and software code that were used during the reasoning pipeline are explained in more detail in the appendices A to D.

7.3.1 Manual curation of the relation types

Most relation types in BioGateway originate from the OBO ontologies. All OBO ontologies exist in BioGateway as RDF graphs, providing the op-

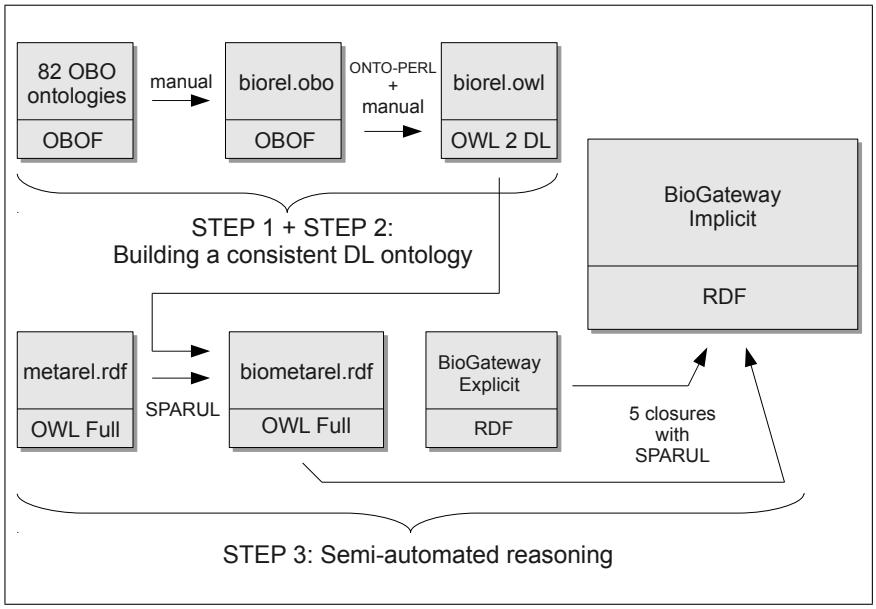


Figure 7.1: A practical implementation of the three-step process for reasoning with bio-ontologies through management of relation semantics. A consistent, validated *biorel.owl* in OWL 2 DL contains all the relation types. It is the starting point for applying 5 important closure rules with a basic RDF tool like SPARQL/Update (SPARUL).

portunity to transform the relational information available in BioGateway with RDF tools. However, standard RDF conversion tools do not properly translate all information embedded in OBO ontologies to RDF, so we initiated the work with the original OBO files for a more expressive translation. All Typedef sections for relation types were separated from all OBO files and through a process of manual curation this long list was reduced to a single valid, consistent OBO file. Text sorting operations and spreadsheets were used to compare and select the best annotated and authoritative relation type entries among the duplicates. In this manner 833 relation type entries were reduced in a consistent effort to 365 unique, curated relation types. The resulting OBO file, *biorel.obo*, is available for download at http://www.semantic-systems-biology.org/metarel/biorel.

The most crucial step in the curation process, central to the decision of using the Metarel/RDF framework, was to make a consistent interpretation of the relation types as either object properties (relation types

between instances) or all-some relation types between classes. Every relation type used between two terms in an OBO file was interpreted as a metarel:AllSomeClassRelationType and the corresponding relation type in the Typedef section as an owl:ObjectProperty. Relation types that were annotated as 'metadata-tags' in the Typedef section were always interpreted as an owl:AnnotationProperty. This interpretation is entirely consistent with the current standardized practices of translation between OBO and OWL DL (OboInOwl) [40], but it is not consistent with the original interpretation that is still commonly held by many OBO ontology developers. Judging from definitions and tags in OBO's Typedef section, the relation types are most often still viewed as class relation types.

As a consequence, some tags that were introduced in an ad hoc manner to solve this ambiguity, like 'inverse_of_at_instance_level' and 'instance_level_is_transitive', were replaced by the standard variants that are captured better by OBO translation tools.

Six relation types, like *has taxonomic rank*, *is valid for taxon* and *is extinct*, have OBO's metadata-tag because they cannot be given an interpretation as object properties. We added this tag to *is integral part of* for the same reason. Its semantics is clear as it can always be written as a combination of the two all-some relation types *is part of* and *has part* in opposite directions: if 'A *is integral part of* B', then every A *is part of* some B and every B *has part* some A. It is interpreted as a *metarel:InvertibleRelationType*, which enables some additional closure rules. However, it does not fit into the general system and it needs to be translated into an annotation property for validation in OWL 2 DL.

A consistent naming system was created, by giving every relation type a name that contains a verb in the third person singular. For instance the name *after* was replaced by *exists after*, as this seemed to be the intended meaning.

Rules that were only poorly formulated as informal comments were upgraded to sound logic. For instance, the comment that any *starts at end of* implies an *is preceded by* was easily translated into OBO's logic by modeling the former as a subproperty of the latter. The new OBO tag 'holds_over_chain' for creating property chains was exploited to its fullest extent and it was added in several cases. For instance, *is directly preceded by* holds over a chain of *has start* and *is end of*. The 'transitive_over' tag became superfluous through the use of 'holds_over_chain'.

One informally asserted rule stated: "Gt influences P & Gt variant_of

G => G influences P". This is a chain rule with one object property in the inverse direction. Interpreting the object properties as all-some relation types between classes we will have *is variant of* from the some-side to the all-side, which does not result in a sound rule on the class level. Indeed, as every Gt is a variant of some entity, it would follow that every entity (everything) influences some P. Implementing such a rule for classes would corrupt the whole knowledge base. The rule was translated into a formal chain rule by using an inverse relation. As no inverse was tagged for *is variant of*, the following choice was made: *is influenced by* holds over a chain of *is influenced by* and *is variant of*. This chain goes from the all-side to the some-side and it retains the intended semantics.

Transitivity was added for all the object properties that were tagged as the inverse of a transitive object property. For instance *is preceded by* was provided with transitivity by the transitivity of *precedes*.

Apart from the names, some dozens of OBO tags for the semantics of relation types had to be altered. Contradictions were nowhere found and the intended semantics could always be retrieved by informal expressions in the comment section of the OBO format and by the way the relation types were actually used in the ontologies.

7.3.2 Translation to the Semantic Web

The use of the available Semantic Web tools for inferring and querying requires a translation to a Semantic Web language. The current standards are OWL and RDF. BioGateway, an RDF store, does not contain any of the OWL profiles. By using Metarel/RDF as a target framework for the translation, we are, however, still using the standards, because Metarel is valid OWL Full and apart from using class relation types, Metarel is fully compatible with the language constructs used in OWL. Moreover, Metarel being valid OWL Full is technically equivalent with it being valid RDF (http://www.w3.org/TR/owl-ref/). Unlike OWL Full, however, Metarel can connect the class relation types that reside in the RDF of BioGateway with the object properties in Biorel.

We translated *biorel.obo* first into *biorel.owl* using ONTO-PERL and adjusted the translated file with some manual curation, which resulted in a valid OWL 2 DL ontology file for the relation types. We added also the chains of relation types, a feature novel in OWL 2. In principle, *biorel.owl* should contain all the expressivity for the rules in Section 7.2.2, even in RDF.

For the purpose of reasoning in BioGateway, *biorel.owl* was not practically useful yet, because it contained only object properties, while BioGateway contained only class relation types. We uploaded *biorel.owl* into the relation meta-graph *metarel.rdf* alongside the other RDF ontologies in BioGateway. The merged graph is called *biometarel* and it is this graph that is used for the reasoning process. With SPARUL updates in the biometarel graph we could connect the object properties of biorel with the class relation types of BioGateway and propagate the semantic rules, like transitivity and chains, to the level of classes.

7.3.3 Inferring new knowledge statements

Each of the five rules that are required for a query system, as discussed in Section 7.2.2, corresponds to a single SPARUL/Update query type. These update queries range over BioMetarel and the ontology graphs in BioGateway. They need to be operated in a recursive loop until there is no new knowledge statement left that can be inferred. The update query for inferring reflexive relations in the human disease graph is shown as an example here:

```
BASE <http://www.semantic-systems-biology.org/>
PREFIX metarel:<http://www.metarel.org/>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
INSERT INTO GRAPH <human_disease_tc> {
  ?Class ?RelType ?Class.
}
WHERE {
  GRAPH <human_disease_tc> {
    ?Class rdf:type ?Type.
  }
  GRAPH <biometarel> {
    ?RelType rdf:type metarel:ReflexiveRelationType.
    ?RelType rdf:type metarel:ClassRelationType.
  }
}
```

A PERL script parametrizes the 5 SPARQL/Update query types with about 2000 graph names. All the inferences from such small graphs are merged later in the large SSB_tc graph through other update queries. This practice implies that the entailment of the inferred triples is fully materialized on a hard disk and when this is executed on BioGateway with OpenLink Virtuoso 5.0.8, it shows an acceptable performance. It takes approximately 20 hours to produce 158 million inferred knowledge statements,

which is reasonable compared to an uploading time of 5 hours for the 401 million original triples. Tens of thousands of SPARQL/Update queries were run during the reasoning phase, since the closure rules, apart from reflexivity, need to run several times on each of the 2000 graphs. Reflexivity, which is executed first, cannot create a part of a chain that triggers the inference of new triples from other closure rules. The priority over *is_a*, the super-relations and the chains, can all fill a missing gap in a chain of transitive relations, which is why they were run at least three times in a fixed order.

Graphs that did not target each other were only integrated after the reasoning phase. Some closure rules, especially the priority over *is_a*, required too much computational performance for running over a large, integrated graph. The priority over *is_a* being computationally the most challenging closure rule, can be understood by realizing that also the *is_a* relation is considered as having priority over itself. In other words, the priority over *is_a* also executes the transitivity of *is_a*. The computational demand for the transitive closure is therefore only a fraction of the demand for the priority over *is_a*.

Not all triples are knowledge statements with a meaningful relation type as a connector. Many triples are used for asserting names, synonyms, definitions, textual annotations, literature references, etc. As these triples are often more verbose, we will call them *verbose triples* as opposed to knowledge statements. For the Gene Ontology we get the following numbers: 54 718 explicit knowledge statements, 643 384 verbose triples and 2 031 247 newly inferred knowledge statements. This implies a multiplication factor of 38,12 for the number of knowledge statements, but a multiplication factor of only 3,90 for the total number of triples. For the complete BioGateway we have a multiplication factor of only 1,39.

The relatively low multiplication factor and the high percentage of verbose triples clearly show that a full materialization does not pose many extra storage-related problems for bio-ontologies. It makes no sense to start the reasoning process in a temporal memory only after a query is launched. It takes 20 hours to generate all the informative knowledge statements, whereas the majority of typical biologically relevant queries take no more than some seconds to produce an answer. A quick response time is absolutely required in the knowledge exploration phase that precedes a more systematic investigation of a new hypothesis. Therefore the materialization of inferred triples is the preferred practice for bio-ontologies.

7.4 Results

We inferred about 158 million knowledge statements through semi-automated reasoning within BioGateway. Most of the time, the inferences are sound within the intuitive system of all-some relations (an exception: plural forms in 'Every *Mammalia* is some *cellular organisms*', following from an archaic system for defining terms in the NCBI Taxonomy) and they can be accessed directly for any term through BioGateway's most basic lookup query. Most of the inferences are rather trivial if they are considered as a single statement, however, their effect becomes clear to those who are querying the knowledge base. Without the inferences, certain queries either simply return only a fraction of the answers potentially available in the knowledge sources, or they require a lot of very specific knowledge on the architecture of the ontologies in the KB to retrieve full results. The reasoning process would need to be done by the query builder and the resulting queries would become huge and slow.

By precomputing all the inferences, the hardest part of the reasoning process happens only once in a single although substantial computational effort, but the results are stored and are available for all subsequent queries. The query builder can now concentrate solely on the intended meaning of the relation types that are used, instead of reconstructing this meaning by his query. He can query over the explicit knowledge statements as well as over the implicit ones.

Queries on the RDF graphs with inferred knowledge statements are now short and the answers are more informative and complete. Imagine a cancer researcher who investigates the ASPP1-proteins (Apoptosis stimulating of p53-protein 1) and she finds in direct manual annotations from GOA that proteins of this type are located in the nucleus and in the cytoplasm and that they participate in the processes 'induction of apoptosis' and 'negative regulation of cell cycle'. Now she would like to see which other proteins fulfill these conditions within mammals.

This query involves just a pattern of knowledge statements in the triple form:

- my_subject *is located in* cytoplasm
- my_subject *participates in* apoptosis
- my_subject *participates in* negative regulation of cell cycle
- my_subject *has source* mammal

The query will return all the classes of biological entities (proteins in this case) that can, within BioGateway, be inferred to be located in the cytoplasm, to participate in apoptosis, etc., instead of searching only through the knowledge statements that were once annotated explicitly by an ontology engineer or a manual curator. 'Mammal' is too generic to be chosen as an annotation for a source species. Also 'cytoplasm', 'apoptosis' and 'negative regulation of cell cycle' have many sublocations and subprocesses that are often chosen for annotations. The query returns 36 types of proteins that actually fulfill all the conditions, but just 29, 29 and 17 respectively if only the explicit annotations are queried for 'cytoplasm', 'apoptosis' and 'negative regulation of cell cycle'. We still get 13 protein types for explicit annotations on all these three conditions, but none for direct annotations on 'mammal'. The relatively high numbers of explicit annotations are due to the fact that they are abundant and redundant in meaning, though taken together still incomplete.

The necessity of each of the five closure rules was investigated separately by recreating the inferred version of BioGateway five times and omitting one of the rules during each recreation. We detected that many inferences followed from several different closure rules, which raises the question whether one of the closure rules can be omitted at all. However, for all 5 recreations of the implicit KB, some of the inferences were missing, indicating that all the closure rules have an essential function. Four specific biological queries in BioGateway illustrate the practical relevance of each of the closure rules. Queries Bio4 and Bio5 are part of the library of preconstructed queries in BioGateway:

- **Query 1:** Which are all the biological processes in which a given protein (dnaJ in *Chlamydophila felis* Fe/C-56) is involved, which are all the other proteins that participate in these biological processes and which cellular locations were annotated for these other proteins? (Bio4 in BioGateway)

- **Query 2:** Which are the proteins that have both the nucleus and the endoplasmatic reticulum as inferred locations, compared and ordered for all the organisms in the KB? (Bio5 in BioGateway)

- **Query 3:** What are the subparts of liver parenchyma?

- **Query 4:** Which are the developmental stages preceding the unfertilised egg stage, and that are themselves preceded by oogenesis stage S6 (the stage during which follicle cell division ceases)?

For each query the number of answers was counted, rendered by either the KB with only the explicit knowledge, on the KB with all the additional inferred knowledge and in each of the KBs that lacked one specific type of closure. The results can be viewed in Table 7.1.

To demonstrate that the additional answers also make biological sense, we will analyze the queries and the corresponding parts of the KB. Query 1 asks for proteins that are involved in the same biological process as a given process. This means that a protein involved in a subprocess is also a good answer. The query asks generically for proteins in the same process and/or the subprocesses, but without the reflexive closure proteins annotated with the exact same process are disregarded (79 answers). Without any closure we get only the proteins annotated with the subprocesses on the level immediately below the original process, but not subprocesses of subprocesses (2 answers). Query 2 fails to return proteins that are annotated with sublocations of the nucleus and the endoplasmic reticulum when either the priority over *is_a* or the chain closure is omitted. This depends on the particular engineering of the Gene Ontology. We find almost exclusively the *is_a* relation type below the nucleus and the endoplasmic reticulum, with only nuclear part *is part of* nucleus and endoplasmic reticulum part *is part of* endoplasmic reticulum, for instance 'germ cell nucleus *is a* nucleus' and 'ARC complex *is a* nuclear part'. The priority over *is_a* propagates *is located in* over all these *is_a*'s, but we need a specific chain rule to propagate *is located in* over *is part of*. The priority over *is_a* generates extra answers for annotations with terms like 'germ cell nucleus' (616 answers). But as no protein was annotated with nuclear part nor endoplasmic reticulum part, we get only the explicit annotations on nucleus and endoplasmic reticulum if the priority over *is_a* is omitted (593 answers). Only if both the chain closure and the priority over *is_a* are in place, proteins with annotations in the hierarchy below nuclear part and endoplasmic reticulum part are retrieved (738 answers). Query 3 requires the transitive closure of *is part of* for finding the subparts of 'liver parenchyma'. Without any closures only 'liver lobule', 'portal lobule' and 'portal triad' are retrieved (3 answers), but not the 6 more specific terms like 'bile canaliculus', which are subparts of the liver lobule and the portal triad. Reflexivity acknowledges that a liver parenchyma is also part of itself (4 answers). Query 4

	Exp.	Imp.	R1	R2	R3	R4	R5
Query 1	**2**	118	**79**	118	118	118	118
Query 2	**593**	738	738	738	**593**	738	**616**
Query 3	**3**	10	**9**	**4**	10	10	10
Query 4	**0**	19	19	**0**	19	**0**	19

Table 7.1: The number of answers to queries compared on explicit knowledge (Exp.) and implicit knowledge (Imp.), and on partial closures where rules for reflexivity (R1), transitivity (R2), priority over subsumption (R3), super-relations (R4) and chains (R5) were omitted respectively. The bold typeface indicates partial loss of logically correct results.

asks for a series of developmental stages. The ontology of developmental stages uses *starts at end of* as a relation type to connect subsequent stages. *Starts at end of* is a subrelation of the transitive relation type *is preceded by*. That is why only answers are found if both the transitive closure and the super-relation closure are implemented (19 answers).

The results show that every query executed in BioGateway that uses any of the 365 relation types in *biorel.obo* benefits from the reasoning process that has created the inferences. The answers to such a query are complete and they correspond to the logical meaning of the relation types as intended by the ontology engineers. This meaning no longer needs to be simulated in the queries.

7.5 Conclusion

With this work, we have chosen to enable efficient querying with the most basic semantic features, instead of hampering the query system with advanced, fully automated reasoning. The division of the reasoning approach in three steps has the benefit that the consistency of the knowledge base is more easily achieved after the execution of the third step. Manipulations of the contents the knowledge base, like insertions, deletions and updates, are left to step 2, which is done by ontology engineers who are maintaining the consistency of their ontology. Also the addition of new rules, which belongs to step 1, is done by external engineers. Step 3 requires a periodical integration effort that relaxes many rules and that assumes consistency of the ontologies that are integrated. After the complete execution of step 3,

the knowledge base is supposed to remain entirely stable for some time.

The precomputing of relational closures appears to be the only way to query on logically inferred statements with SPARQL. There is some limited support for reasoning with transitivity in SPARQL, but this makes the query harder to write, slower to compute and it does not provide support for other logical features. RDF triple store developers should provide the option to define a set of SPARUL rules that are operated recursively during load time and loaded in a separate graph with a similar name as the graph with the original contents. This enables to launch the same queries easily with or without inferences in place, which is currently supported by the OWLIM triple store.

Many different ontology engineers have collaborated in the coordinated development of more than 80 OBO ontology files. We have brought consistency to the stack of relation types in these files by gathering all the relation types in Biorel and translating them to OWL 2 DL. After merging the OWL translation of Biorel with Metarel in the RDF store BioGateway, we could infer 158 million previously hidden knowledge statements from the explicitly asserted knowledge in the OBO ontologies, GOA annotations for about 2000 species, UniProtKB/Swiss-Prot and the NCBI Taxonomy. The inferred knowledge statements can be used for biological hypothesis generation through querying. The success of our methodology is due to the soundness of the OBO ontologies, the use of Semantic Web tools and the semi-automated approach of reasoning.

This work shows that a small set of simple rules for bio-ontologies results in efficient practices for reasoning and querying. As many researchers are involved in building bio-ontologies, more restrictive guidelines and principles for building bio-ontologies are required in order to obtain more uniformity and reach more convergence for knowledge management in the Life Sciences.

Chapter 8

Conclusions

The total amount of available knowledge and data in the Life Sciences has been growing exponentially during the last decade. This is caused by new techniques for sequencing and analyzing the genome and by an exponential growth of scientific literature in the biomedical domain. These rapid evolutions call for knowledge management in the Life Sciences with the most modern techniques in computer science.

Computer science ontologies and the Semantic Web are cornerstones for modern knowledge management. Many biomedical resources were developed along the lines of these paradigms in the last decade, for which the OBO Foundry [37] has been a driving force and OWL [108] a key technology. Literature resources like PubMed that have been growing exponentially have been taken care of by text-mining applications, which have often used ontologies as a semantic framework [58].

These modern technologies are based on two old scientific domains that are rooted in philosophy: Logic and Ontology. The field of Logic went through a series of developments during the last 150 years, which has converted it from very ambiguous philosophic discourse towards strict mathematics and later towards computer science. Ontology seems to be going in the same direction during the last 15 years. The popularity of ontology languages like OWL has attracted much attention from computer scientists and application-oriented knowledge engineers towards the philosophic theories about the representation of reality. This has given a strong impetus to the development of comprehensive, reality-based ontologies like BFO [91]. OWL is the ultimate combination of Logic and Ontology, since it is designed to model ontologies in strictly decidable DLs [131].

The Semantic Web has built a bridge from the modern basis of computer science, like IRIs [102] for identification, HTTP [96] for Internet transfer, XML [103] for syntactical representation and Unicode [101] for symbols, towards knowledge management in OWL. The most important technology that constitutes this bridge is the graph language RDF [104], that can be queried through SPARQL [106]. It provides a graph model for OWL, which enables SPARQL queries about OWL ontologies. These queries can either be logically sound in the sense of DL, or they can query for modeling choices, like direct superclasses or closest common superclasses, that cannot be expressed as a DL query. Users of ontologies need both types of queries, which declares the choice for representing OWL in RDF. The bridge is completed by the syntactical representation of RDF in XML. The whole stack of technologies has become known as the Semantic Web Stack.

RDF is slowly becoming the engine of the Semantic Web. Related technologies, like SPARUL [126], SPIN [141], federated querying [144] and Linked Data [143], are currently being developed. SPARUL provides SPARQL-based rules for updating RDF, which is complemented by SPIN for writing SPARQL queries as RDF. The combination of both technologies, SPARUL and SPIN, facilitates RDF stores that contain their management and reasoning rules internally. Federated querying exploits the available resources, both storage and computing resources, for executing SPARQL queries worldwide.

Linked Data is the latest Semantic Web paradigm that promotes several ideas like openness of data, browsability of RDF graphs and HTTP-style IRIs. The combination of these ideas might, ideally within the coming years, enable lay users to browse RDF-formatted data through clicking, hovering, and drag-and-drop in web browsers like Firefox.

Contributions of the thesis

Chapter 4 describes the engineering of a library of SPARQL queries, and the benchmarking of the performance of RDF storage systems for query answering through this library. The library of SPARQL queries was designed as a part of a larger system, the DIAMONDS platform, that facilitated queries about the cell cycle for biologists. The platform, which was engineered as a Java application within a browser environment, has shown nice results, was published and delivered to the funding providers, and has enabled us to derive some important lessons about Semantic Web

technologies. Unfortunately, the final design of the platform caused severe instabilities in the browser due to memory overflows deriving from the visualization module. The Semantic Web related modules were reused in the BioGateway project later.

RDF and OWL have worked together successfully for CCO, because RDF was used for querying and OWL for consistency checking. The attempts to query CCO have shown that RDF is perfectly suited for supporting practical applications that browse and retrieve knowledge. Useful queries on CCO in OWL, that cannot be answered in RDF, could not be identified. Querying the RDF version of CCO through SPARQL on the other hand, facilitated queries that addressed the model itself, like retrieving direct super- and subclasses and restricting queries on labels instead of logical class descriptions. SPARQL can return all the classes in the path to the root of an ontology and it can even find the closest common superclass of two classes. E.g. if a biologist wants to see the closest common ancestor of 'human' and 'horse', she expects something like 'mammal'. SPARQL can return this answer, whereas a DL query in OWL should return the class that is the union of all humans and all horses. It has to be acknowledged that the modeling choices of ontology engineers contain a lot of useful information that is obscured by fully automated reasoning.

The development of CCO in RDF and the creation of a library of SPARQL queries for CCO, has enabled to compare the performance of different RDF storage systems. We can conclude that the standards that were developed for RDF and SPARQL were followed rather well by the implementers of these stores. OWLIM has optimized the response time of uncomplicated SPARQL queries on CCO, which consists of millions of triples, into the range of milliseconds. However, some query constructs, like filtering results through regular expressions, were not always handled very optimally by the RDF storage systems.

Chapter 5 introduces the Metarel relation ontology. Metarel supports RDF modeled as MDGs, which can increase the abilities of SPARQL even more. RDF/MDG can complement OWL/RDF, the standard syntax for expressing OWL in RDF. OWL/RDF has the virtue of containing all the information that is expressed in OWL, however, it is not designed to be easily queried with SPARQL. RDF/MDG on the other hand can easily be queried with SPARQL and updated with SPARUL rules, but it is less expressive than DL-based profiles of OWL.

Metarel has roughly three use cases that complement the OWL/RDF

syntax. First of all, inferences created by OWL reasoners that are expressible by Metarel as arcs in an MDG, can be materialized explicitly to make them available for SPARQL querying. Secondly, Metarel can give a stronger ontological commitment to data with respect to the distinction between instances and classes, without changing the data itself. This provides an attractive step towards decidable reasoning with SPARUL (semi-automated) or OWL (fully automated). Thirdly, new data resources could immediately be developed as RDF/MDG, annotated with Metarel, providing a straight-forward conversion to OWL DL.

Linked Data relies on RDF graphs that are browsable. For being browsable, query systems should be able to follow blank nodes automatically. It is not clear how this can be achieved with SPARQL alone, without the need for a procedural language like Java. Since RDF/MDG does not contain any blank nodes, it is highly browsable and it can be queried by SPARQL without any additions. We can conclude that Metarel works very well together with this recent evolution in the Semantic Web.

BioGateway, described in Chapter 6, is a resource that has the RDF/MDG format. It brings the ideas of the Semantic Web into practice by providing a SPARQL query endpoint for biomedical resources like GOA and GO for genome annotations, UniProt for proteins, NCBI Taxonomy for organisms and OBO ontologies for very diverse biomedical domains. The identifiers of the biological terms in these resources are organized within the namespace of SSB. This is sufficient to launch integrated queries over these biomedical resources. UniProt and GOA use the same identifiers for proteins, GOA uses GO for annotations of proteins, GOA and UniProt use the NCBI Taxonomy for the annotation of organisms, and so on. This reuse of terms with the same IRIs is the bottom layer of the Semantic Web, which appears to be very important for BioGateway. As a result, BioGateway can answer complex queries like “Which proteins are involved in disease A and located in cellular component B, that are also interacting with protein C?”

The modeling choices for converting the original biomedical resources into RDF for BioGateway, have been optimized by feedback from querying the resources with SPARQL. This has made BioGateway in accordance with the idea of creating direct RDF-links between the terms that are of relevance, as expressed in Chapter 5 about Metarel. Direct links and the MDG-format are crucial for acquiring a high queryability in BioGateway.

Querying BioGateway was often hampered by serious performance bottlenecks. The fundamental reason for these bottlenecks is the worst-

case complexity of SPARQL, which is PSPACE complete. This means that queries may cost exponential time and polynomial memory space in function of the size of the data. SPARQL queries that use only the AND, the UNION and the FILTER operators, excluding the OPTIONAL operator, are already NP-complete, which is as hard as solving the traveling salesman problem [125]. This makes the optimization of triple store performance a very challenging issue. The only practical solution for SPARQL query engineers is to create queries that are well supported.

BioGateway has assigned new IRIs to the concepts from the ontologies it integrates, by fitting them all in the IRI-namespace of SSB. This is not an ideal practice with respect to integration on an even larger scale, for instance through federated querying. Data providers and the OBO Foundry in particular, have paid more attention to the standardization of IRIs since the development of BioGateway in 2008. For this reason, BioGateway would benefit from an update that uses the more recently developed standards for IRIs, especially for those that stem from the OBO Foundry. On the other hand, it is still not entirely clear to what extent integrated knowledge bases should adopt different IRI schemes from its original sources. For instance, if also the OBO Foundry would accept a different IRI namespace for every of its ontologies, there would be so many namespace prefixes required that querying and integration becomes hard. Continuous discussions on this subject are required in order to reach convergence for recommendations.

Several ideas were tested in BioGateway, like the redundant loading of the same resources in separate RDF graphs and in integrated RDF graphs, the inclusion of MetaOnto, an ontology about ontologies, the creation of transitive closures and the separation of RDF graphs with logical inferences from RDF graphs with the original resources. The resulting architecture of BioGateway has enabled more queries and has decreased their response times.

The redundant loading of resources over separate RDF graphs was done in order to deal better with the slowing down of queries as graph are getting larger. Once an RDF graph was isolated, the responsiveness of the queries on that graph remained more or less stable and acceptable. We can conclude from this that triple stores are very scalable as long as more RDF graphs are created during the growth of the whole store. The addition of extra graphs with logical inferences did not really affect the responsiveness of queries on the original graphs.

Another important idea was the creation of BioMetarel for the integration and the management of relation types in BioGateway. It is not enough that biomedical resources use the same IRIs for biomedical terms, but not for relation types. All the relation types in BioGateway were assembled, made consistent and annotated with terms in Metarel. Apart from facilitating queries, BioMetarel held the promise of facilitating logical reasoning.

This logical reasoning has turned out well with the use of SPARUL, as described in Chapter 7. The reasoning was not achieved in a single step, which is the approach of fully automated reasoning in DL, but in three steps: 1) choosing a DL for representing resources, 2) building the resources while running the DL reasoner on small modules for consistency checking and 3) creating useful inferences for querying in a less expressive, tractable logic, that does not fail due to remaining inconsistencies. The less expressive logic in step 3 was defined as a subset of well-investigated DLs like OWL EL and OWL DL, which could be algorithmically implemented through five closure rules that are easily expressed in SPARUL: reflexivity, transitivity, priority of isa, superrelations and chains.

The work of many others in steps 1 and 2 could be reused by choosing for this approach. They have been building resources and improved their consistency with the use of DL, whereas the creation of inferences for querying only required work in step 3. The quality of ontologies was good enough with respect to the logical consistency of classes in a subsumption hierarchy, apart from the poor naming convention in NCBI, which uses plurals in labels. This level of quality does not apply to the relation types, which were treated too differently in the original resources. The creation and the validation of Biorel in OWL DL, and the annotation of Biorel with Metarel, which resulted in BioMetarel, was work that belonged to steps 1 and 2.

Domains and ranges of relation types are advanced logic constructs that can result in many inconsistencies. They were not considered in step 3 for the creation of inferences, but they were not even included in step 1 for the creation of Biorel. This reflects the fact that the ontological commitment of the original biomedical resources was too different for unifying them in a logic that includes domain and ranges. The developers of BFO are currently working towards a stronger ontological commitment for all biomedical ontologies.

The lessons learned from the reasoning approach, and *biorel.obo* in

particular, were discussed on the mailing lists that are used by the many developers of the OBO ontologies: obo-discuss, obo-relations and bfo-discuss. The differences between *biorel.obo* and *RO.obo*, as well as the translation to OWL, were highlighted in a discussion with Chris Mungall, the main developer of *RO.obo*. Biorel has bent a controversy about the naming of relations in favor of the practice to always use a verb in the third person singular, like *is part of* instead of *part of*. Also the practice to use human readable IRIs for relations, instead of numerical values, was discussed in the context of BFO. Unfortunately, in spite of the positive results with Biorel in BioGateway, and in spite of the flexibility offered by IRIs, current efforts in the BFO community are centered around the development of a small set of non-human readable identifiers that would eventually replace RO. This evolution will certainly not help the development of SPARQL libraries that reveal the contents of OBO ontologies. On the other hand, it provides more flexibility for changing the names for the relations in the future, without obsoleting the identifiers. It also creates more coherence with the understandable practice to use numerical identifiers for large sets of classes in rapidly evolving biomedical domains, like genomics and proteomics. If this practice is compared with IRIs in OWL, we must conclude that the IRIs *rdf:type* and *rdfs:subClassOf* are somewhat outdated, but much more useful than IRIs like *owl:000123* and *owl:000097*. In that sense an updated version of Biorel might fulfill a practical role in the future as a work-around for the flaws in the theoretical solutions.

Also the results of the benchmarking of different RDF stores has raised discussions with the developers of OWLIM, which may eventually lead to better tools. The real contribution of the work in the thesis relates thus to the field of Knowledge Management itself: the testing of semantic technologies and the creation of modeling practices that may lead to better standards. BioGateway, with logical inferences in place, is mostly of use for ontology engineers and Semantic Web programmers. Bioinformaticians may explore the possibilities that it creates, however, RDF stores like BioGateway are not mature enough yet to be directly useful for the target user, the biological expert. Scientists that do not have a background in computer science need clear interfaces that guide them in their knowledge exploration. BioGateway should be evaluated as a building block in the software layers of such a system.

General conclusions

Bio-ontologies and the Semantic Web are two important evolutions for knowledge management in the Life Sciences. They provide a logical framework, universal identifiers and tools for the integration of knowledge. However, in order to become really useful for Life Sciences researchers, both pillars need to mature further and become integrated in the biology toolbox. As the amount of biomedical knowledge keeps growing exponentially, the scalability of Semantic Web tools should be a main concern. Slow queries, intimidating interfaces and immature software form a real obstacle for the exploitation of biomedical knowledge repositories.

The research in this thesis shows new directions and paves the way by providing actual software solutions. This is mostly achieved by engineering new technologies (Metarel and reasoning with SPARUL) and by providing middle-ware (BioMetarel and SPARQL libraries) for end-ware (the DIAMONDS platform and BioGateway).

RDF stands out as the W3C endorsed standard for the Semantic Web that is most promising for the integration of biomedical knowledge. It is complemented by the fully automated solutions provided by OWL, in particular for the consistency checking of ontologies and formally represented knowledge resources. However, when it comes down to querying and creating logical inferences on large-scaled knowledge bases, the Semantic Web Stack relies on RDF and its query language SPARQL. The engineering of Metarel, based on notions of Description Logics on the one hand, but RDF-compatible multidigraphs on the other hand, acknowledges these practical benefits.

In spite of the benefits, the large and diverse possibilities of querying RDF demands better browsing and visualization tools to make the technology more accessible to biologists. Parameterizing and reworking the SPARQL code is still the best option for acquiring all the expressivity of the SPARQL query language. The direct relations between classes used in Metarel and BioGateway may help overcome some of the current shortcomings pertaining to reasoning and browsing.

On the side of the development of bio-ontologies, more efforts are required: ontology engineers should reuse other bio-ontologies to avoid duplication, create appropriate relations, provide identifiers, synonyms, definitions and cross-references. The Semantic Web architecture is perfectly suited for exploiting an orthogonal, cross-linked set of bio-ontologies. BioGateway, with the inferences that were created through Metarel-based rea-

soning, can be used to identify the glitches in bio-ontologies and to improve them further in a consistent manner.

The future perspectives of knowledge management on the Semantic Web depend on recent evolutions like Linked Data and federated querying. The Semantic Web needs to strengthen from the bottom of its stack, the IRIs, to the top. When data providers start to release data systematically with the use of IRIs, that can be put on the Web as Linked Data with little effort, then the standards that were developed higher in the stack, like RDF and OWL, will gain in importance. However, more compatibility will have to be created with classical approaches like relational databases.

Future work

As a future work on BioGateway, the knowledge base would benefit from the inclusion of more data, like biological pathways, for enabling answers to a broader range of interesting questions. A graphical interface that is specifically developed for the library of queries in BioGateway could make the system more attractive for the intended end users. In-depth queries on very specific research questions would provide a test for the biological usefulness of the inferences. For example, the inferences on GO and the NCBI Taxonomy will allow to compare gene functions across species and kingdoms.

The work on Metarel might lead to better standards for the usage of relations in the Semantic Web. It could be of help in bridging Semantic Web relations in their most simple form (as in Linked Data) and relations that express more advanced logical statements (as in OWL and formal Ontology).

Chapter 9

Nederlandstalige samenvatting

Het werk dat in deze thesis beschreven wordt, heeft betrekking op een aantal stappen die gezet zijn in een wetenschappelijk domein dat het beste kan omschreven worden als 'kennisbeheer in de levenswetenschappen'. Dit kennisbeheer is van toenemend belang door toedoen van de snelle vooruitgang die geboekt wordt in de bioinformatica. Nieuwe technieken in de ontrafeling van het DNA en betere computationele verwerking van genoomgerelateerde data hebben ervoor gezorgd dat erg gedetailleerde biomedische kennis in enorme volumes beschikbaar is geworden voor de wetenschap.

Zoals ook uitgelegd wordt in Hoofdstuk 1, zijn de hoofdstukken 2 en 3 inleidend, terwijl de daaropvolgende hoofdstukken, van 4 tot en met 7, de wetenschappelijke stappen beschrijven die door de auteur gezet zijn. Ze houden verband met wetenschappelijke publicaties die gemaakt zijn in internationale tijdschriften en internationale wetenschappelijke conferenties. Hoofdstuk 8 bevat de algemene conclusies die uit de thesis kunnen worden getrokken.

Hoofdstuk 2 geeft een inleiding op kennisbeheer in de levenswetenschappen als wetenschappelijk onderzoeksdomein. Om te beginnen wordt het begrip 'kennis' er onderscheiden van 'data' en 'informatie'. Informatie bestaat uit data en meta-data. Kennis kan beschouwd worden als informatie waarvan de betekenis van een welbepaalde interpretatie is voorzien. De rest van het hoofdstuk beschrijft de enorme inspanningen die in dit domein reeds door anderen werden gezet. Biomedische kennis is doorheen de jaren vervat geweest in allerlei kennisbronnen met soms heel diverse formaten. Waar dit tot voor tien jaar vooral klassieke relationele databanken waren,

maken tegenwoordig ontologieën een sterke opgang. Men heeft de laatste jaren ook veel inspanningen geleverd om kennisbronnen beschikbaar te maken via het zogenaamde 'Semantische Web', dat als een soort computerverstaanbare opvolger van het Wereld Wijde Web gepromoot is geworden. Uiteraard vormen publicaties in natuurlijke taal via tijdschriften en artikels ook nog steeds een belangrijke component van de beschikbare kennis.

Kennismanagement is een onderdeel van de computerwetenschappen en steunt sterk op de ontwikkelingen die zich hebben voorgedaan in de gebieden van de Logica en de Ontologie in de vorige eeuw. In Hoofdstuk 3 wordt daarom een inleiding gegeven op deze wetenschappelijke domeinen, teneinde alle begrippen die in de thesis aan bod komen op een natuurlijke wijze te introduceren. Begrippen als 'ontologie', 'logische taal', 'instance' (voorbeeld), 'class' (klasse), 'decidability' (beslisbaarheid) en 'tractability' (praktische berekenbaarheid), krijgen daardoor een betekenis. Het hoofdstuk sluit af met de moderne technologieën, zoals SPARQL, RDF en OWL, die een basis vormen voor het wetenschappelijk werk in deze thesis. In een persoonlijke analyse wordt uitgelegd dat OWL zeer geschikt is voor het nagaan van de logische consistentie van ontologieën, terwijl RDF meer geschikt is voor het bevragen ervan.

Hoofstuk 4 beschrijft de ontwikkeling van een bevragingslaag voor de Celcyclus Ontologie (Cell Cycle Ontology, CCO). Het DIAMONDS platform is een online software-dienst die gebruikers met specifieke biologische vragen in verband met de celcyclus in staat stelt om grafisch doorheen de gedetailleerde kennis in CCO te zoeken. Deze laag omvat een lijst bevragingstypes (of queries) die zowel elementaire ontologie-gerelateerde opzoekingen doet, zowel als meer geavanceerde biologische opzoekingen. Het DIAMONDS platform is later vervangen geworden door BioGateway ('gateway' betekent doorgang of tunnel), die ook op deze bevragingslaag kon verderbouwen.

Een tweede luik in hoofstuk 4 behandelt de vergelijking van de berekeningssnelheden van verschillende softwareprogramma's die RDF opslagsilo's kunnen bevragen. De RDF-versie van CCO en de lijst met bevragingstypes die was ontwikkeld voor CCO, zijn daarvoor als ijkingsstandaard gebruikt geworden. Samen vormen ze de NTNU ijkingsstandaard (NTNU benchmark).

In Hoofdstuk 5 wordt Metarel behandeld, een meta-ontologie voor relaties in biomedische ontologieën. De uitgebreide mogelijkheden die RDF biedt om logische verbanden te modelleren, hebben het bevragen van

RDF-geformatteerde kennisbronnen met behulp van SPARQL bemoeilijkt. Metarel biedt daarom de mogelijkheid om relaties in een eenvoudiger RDF-model van een logische betekenis te voorzien. Een dergelijk eenvoudiger model stemt overeen met een multidigraaf (MDG). In tegenstelling tot de bestaande technologieën voor het Semantische Web, kunnen met Metarel rechtstreekse relaties tussen klassen logisch beschreven worden.

BioGateway is een geïntegreerde, RDF-gebaseerde kennisbasis waarin geavanceerde opzoekingen mogelijk zijn over biomedische kennis. Haar architectuur en implementatie, beschreven in Hoofdstuk 6, bouwt verder op de voorgaande hoofdstukken. De bevragingslaag die ontwikkeld was in Hoofdstuk 4 is verder op punt gezet en heeft als basis gediend om de architectuur van BioGateway te optimalizeren. Metarel heeft in BioGateway haar eerste concrete toepassing gevonden. Ze werd er samengevoegd met Biorel, die een lijst van biomedische relaties bevat. Hun product vormt BioMetarel, dat instaat voor het beheer van de relaties in BioGateway. Een andere meta-ontologie, MetaOnto, werd ontwikkeld voor de bevraging en het beheer van meta-informatie over de kennisbronnen die in BioGateway werden geïntegreerd.

Hoofdstuk 7 levert een bewijs dat de aanpak die in de vorige hoofdstukken werd ontwikkeld werkt. Metarel is er gebruikt geworden om enkele honderden miljoenen nieuwe relaties te leggen tussen de begrippen in BioGateway, op basis van logische regels die geformuleerd werden met SPARQL/Update. Door te werken met directe relaties tussen klassen, vervat in een enkele RDF triple met een onderwerp, een predikaat en een voorwerp, is BioGateway heel aantrekkelijk gebleven om te bevragen via SPARQL.

Tenslotte levert Hoofdstuk 8 een aantal algemene conclusies die uit het werk in deze thesis kunnen getrokken worden voor het bredere wetenschappelijke domein van kennisbeheer in de levenswetenschappen. Twee belangrijke evoluties zijn daarin van belang gebleken: het Semantische Web en bio-ontologieën. Beide steunpilaren moeten verder rijpen en nog meer opgenomen worden als standaardtechnieken in de software-omgeving van biologische wetenschappers. Deze thesis heeft nieuwe methoden onderzocht om deze doelen te bereiken. In het bijzonder is het nut van de Semantische Web taal RDF en en haar bevragingstaal SPARQL voor de praktische bevraging van biomedische kennisbronnen eruit gebleken. Metarel heeft deze tak van het Semantische Web nog verder ondersteund door multidigrafen in RDF van een logische betekenis te voorzien, naar analogie van de Beschrijvingslogica (Description Logics) die gebruikt wordt in

de Web Ontologie Taal (Web Ontology Language, OWL). Daardoor werd het ook mogelijk om bio-ontologieën aan een praktische test te onderwerpen. De nieuwe relaties die in BioGateway gevormd werden via de logische regels in Metarel, kunnen gebruikt worden om onvolkomenheden in bio-ontologieën na te speuren en ze in de toekomst door een beter ontwerp te vermijden.

Bibliography

[1] E. Antezana, M. Egaña, B. De Baets, M. Kuiper, and V. Mironov, "ONTO-PERL: An API for supporting the development and analysis of bio-ontologies," *Bioinformatics*, vol. 24, no. 6, pp. 885–887, 2008.

[2] E. Antezana, M. Egaña, W. Blondé, A. Illarramendi, I. Bilbao, B. De Baets, R. Stevens, V. Mironov, and M. Kuiper, "The Cell Cycle Ontology: An application ontology for the representation and integrated analysis of the cell cycle process," *Genome Biology*, vol. 10, no. 5, pp. R58+, 2009.

[3] V. Mironov, E. Antezana, M. Egaña, W. Blondé, B. De Baets, R. Stevens, and M. Kuiper, "Flexibility and Utility of the Cell Cycle Ontology," *Applied Ontology*, vol. 6, no. 3, pp. 247–261, 2011.

[4] V. Mironov, N. Seethappan, W. Blondé, E. Antezana, A. Splendiani, and M. Kuiper, "Gauging triple stores with actual biological data," *BMC Bioinformatics*, vol. 13, no. Suppl 1, pp. S3+, 2012.

[5] W. Blondé, E. Antezana, B. De Baets, V. Mironov, and M. Kuiper, "Metarel: an Ontology to support the inferencing of Semantic Web relations within Biomedical Ontologies," in *Proc. of the International Conference on Biomedical Ontologies (ICBO)*, pp. 79–82, 2009.

[6] E. Antezana, W. Blondé, M. Egaña, A. Rutherford, R. Stevens, B. De Baets, V. Mironov, and M. Kuiper, "Structuring the life science resourceome for semantic systems biology: lessons from the BioGateway Project," *SWAT4LS, Burger A, Paschke A, Romano, et al, eds*, vol. 435, 2008.

[7] E. Antezana, W. Blondé, M. Egaña, A. Rutherford, R. Stevens, B. De Baets, V. Mironov, and M. Kuiper, "BioGateway: a semantic systems biology tool for the life sciences.," *BMC Bioinformatics*, vol. 10 Suppl 10, no. Suppl 10, pp. S11+, 2009.

[8] W. Blondé, V. Mironov, A. Venkatesan, E. Antezana, B. De Baets, and M. Kuiper, "Reasoning with bio-ontologies: using relational closure rules to enable practical querying," *Bioinformatics*, vol. 27, no. 11, pp. 1562–1568, 2011.

[9] D. Bostock, *Plato's Theaetetus*. Oxford University Press, USA, 1991.

[10] E. Antezana, M. Kuiper, and V. Mironov, "Biological knowledge management: the emerging role of the Semantic Web technologies," *Briefings in Bioinformatics*, vol. 10, no. 4, pp. 392–407, 2009.

[11] C. Linnaeus, *Systema naturae per regna tria naturae, secundum classes, ordines, genera, species, cum characteribus differentiis, synonymis, locis*. 1st ed., 1735.

[12] R. A. Drysdale, M. A. Crosby, and , "FlyBase: genes and gene models.," *Nucleic Acids Res*, vol. 33, no. Database issue, 2005.

[13] C. Béroud and T. Soussi, "The UMD-p53 database: New mutations and analysis tools," *Human Mutation*, vol. 21, no. 3, pp. 176–181, 2003.

[14] C. J. Carter, "Convergence of genes implicated in Alzheimer's disease on the cerebral cholesterol shuttle: APP, cholesterol, lipoproteins, and atherosclerosis," *Neurochemistry International*, vol. 50, no. 1, pp. 12–38, 2007.

[15] C. Munthe, "The use of human biobanks. Ethical, social, economical, and legal aspects," *Journal of Medical Ethics*, vol. 29, no. 2, p. 123, 2003.

[16] B. Verslyppe, B. Slabbinck, W. De Smet, P. De Vos, B. De Baets, and P. Dawyndt, "StrainInfo.net Web services: enabling microbiologic workflows such as phylogenetic tree building and biomarker comparison," in *IEEE Fourth International Conference on eScience*, (Piscataway, NJ, USA), pp. 603–7, 2008.

[17] R. Kaaks, E. Lundin, J. Manjer, S. Rinaldi, C. Biessy, S. Soderberg, P. Lenner, L. Janzon, E. Riboli, G. Berglund, and G. Hallmans, "Prospective study of IGF-I, IGF-binding proteins, and breast cancer risk, in Northern and Southern Sweden," *Cancer Causes & Control*, vol. 13, no. 4, pp. 307–316, 2002.

[18] S. Miyazaki, H. Sugawara, K. Ikeo, T. Gojobori, and Y. Tateno, "DDBJ in the stream of various biological data.," *Nucleic Acids Res*, vol. 32, no. Database issue, 2004.

[19] G. Stoesser, W. Baker, A. van den Broek, E. Camon, M. Garcia-Pastor, C. Kanz, T. Kulikova, R. Leinonen, Q. Lin, V. Lombard, R. Lopez, N. Redaschi, P. Stoehr, M. A. A. Tuli, K. Tzouvara, and R. Vaughan, "The EMBL nucleotide sequence database.," *Nucleic acids research*, vol. 30, no. 1, pp. 21–26, 2002.

[20] D. A. Benson, I. Karsch-Mizrachi, D. J. Lipman, J. Ostell, and D. L. Wheeler, "GenBank.," *Nucleic acids research*, vol. 36, no. Database issue, pp. D25–D30, 2008.

[21] A. Bairoch, R. Apweiler, C. H. Wu, W. C. Barker, B. Boeckmann, S. Ferro, E. Gasteiger, H. Huang, R. Lopez, M. Magrane, M. J. Martin, D. A. Natale, C. O'Donovan, N. Redaschi, and L.-S. L. Yeh, "The universal protein resource (UniProt)," *Nucleic Acids Research*, vol. 33, no. suppl 1, pp. D154–D159, 2005.

[22] N. Hulo, A. Bairoch, V. Bulliard, L. Cerutti, E. De Castro, P. S. Langendijk-Genevaux, M. Pagni, and C. J. Sigrist, "The PROSITE database.," *Nucleic acids research*, vol. 34, no. Database issue, 2006.

[23] D. Maglott, J. Ostell, K. D. Pruitt, and T. Tatusova, "Entrez gene: gene-centered information at NCBI.," *Nucleic Acids Res*, vol. 33, no. Database issue, 2005.

[24] K. D. Pruitt and D. R. Maglott, "RefSeq and LocusLink: NCBI gene-centered resources," *Nucleic Acids Res*, vol. 29, no. 1, pp. 137–140, 2001.

[25] G. Parra, E. Blanco, and R. Guigo, "GeneID in drosophila," *Genome Res*, vol. 10, no. 4, pp. 511–515, 2000.

[26] M. Kanehisa and S. Goto, "KEGG: Kyoto encyclopedia of genes and genomes," *Nucleic Acids Research*, vol. 28, no. 1, pp. 27–30, 2000.

[27] H. Hermjakob, L. Montecchi-Palazzi, C. Lewington, S. Mudali, S. Kerrien, S. Orchard, M. Vingron, B. Roechert, P. Roepstorff, A. Valencia, H. Margalit, J. Armstrong, A. Bairoch, G. Cesareni, D. Sherman, and R. Apweiler, "IntAct: an open source molecular interaction database.," *Nucleic acids research*, vol. 32, no. Database issue, 2004.

[28] G. Joshi-Tope, M. Gillespie, I. Vastrik, P. D'Eustachio, E. Schmidt, B. de Bono, B. Jassal, G. R. Gopinath, G. R. Wu, L. Matthews, S. Lewis, E. Birney, and L. Stein, "Reactome: a knowledgebase of biological pathways.," *Nucleic acids research*, vol. 33, no. Database issue, pp. D428–432, 2005.

[29] K. Spackman and K. Campbell, "Compositional concept representation using SNOMED: Towards further convergence of clinical Terminologies," *Journal of the American Medical Informatics Association*, no. S, pp. 740–744, 1998.

[30] S. Schulz, R. Cornet, and K. Spackman, "Consolidating SNOMED CT's ontological commitment," *Applied Ontology*, vol. 6, no. 1, pp. 1–11, 2011.

[31] M. Ashburner, C. A. Ball, J. A. Blake, D. Botstein, H. Butler, J. M. Cherry, A. P. Davis, K. Dolinski, S. S. Dwight, J. T. Eppig, M. A. Harris, D. P. Hill, L. Issel-Tarver, A. Kasarskis, S. Lewis, J. C. Matese, J. E. Richardson, M. Ringwald, G. M. Rubin, and G. Sherlock, "Gene ontology: tool for the unification of biology. The Gene Ontology Consortium.," *Nature genetics*, vol. 25, no. 1, pp. 25–29, 2000.

[32] E. Camon, M. Magrane, D. Barrell, D. Binns, W. Fleischmann, P. Kersey, N. Mulder, T. Oinn, J. Maslen, A. Cox, and R. Apweiler, "The gene ontology annotation (GOA) project: implementation of GO in SWISS-PROT, TrEMBL, and InterPro.," *Genome research*, vol. 13, no. 4, pp. 662–672, 2003.

[33] C. Rosse and J. L. V. Mejino, "A reference ontology for bioinformatics: The foundational model of anatomy," *Journal of Biomedical Informatics*, vol. 36, no. 36, pp. 478–500, 2003.

[34] P. de Matos, A. Dekker, M. Ennis, J. Hastings, K. Haug, S. Turner, and C. Steinbeck, "ChEBI: a chemistry ontology and database," *Journal of Cheminformatics*, vol. 2, no. Suppl 1, pp. P6+, 2010.

[35] B. Trombert-Paviot, J. M. Rodrigues, J. E. Rogers, R. Baud, E. van der Haring, A. M. Rassinoux, V. Abrial, L. Clavel, and H. Idir, "GALEN: a third generation terminology tool to support a multi-purpose national coding system for surgical procedures.," *Int J Med Inform*, vol. 58-59, pp. 71–85, 2000.

[36] E. W. W. Sayers, T. Barrett, D. A. A. Benson, S. H. H. Bryant, K. Canese, V. Chetvernin, D. M. M. Church, M. Dicuccio, R. Edgar, S. Federhen, M. Feolo, L. Y. Y. Geer, W. Helmberg, Y. Kapustin, D. Landsman, D. J. J. Lipman, T. L. L. Madden, D. R. R. Maglott, V. Miller, I. Mizrachi, J. Ostell, K. D. D. Pruitt, G. D. D. Schuler, E. Sequeira, S. T. T. Sherry, M. Shumway, K. Sirotkin, A. Souvorov, G. Starchenko, T. A. A. Tatusova, L. Wagner, E. Yaschenko, and J. Ye, "Database resources of the national center for biotechnology information.," *Nucleic Acids Research*, vol. 37, no. Database issue, pp. D5–15, 2009.

[37] B. Smith, M. Ashburner, C. Rosse, J. Bard, W. Bug, W. Ceusters, L. J. Goldberg, K. Eilbeck, A. Ireland, C. J. Mungall, N. Leontis, P. Rocca-Serra, A. Ruttenberg, S.-A. Sansone, R. H. Scheuermann, N. Shah, P. L. Whetzel, and S. Lewis, "The OBO foundry: coordinated evolution of ontologies to support biomedical data integration," *Nature Biotechnology*, vol. 25, no. 11, pp. 1251–1255, 2007.

[38] B. Smith, W. Ceusters, B. Klagges, J. Kohler, A. Kumar, J. Lomax, C. Mungall, F. Neuhaus, A. Rector, and C. Rosse, "Relations in biomedical ontologies," *Genome Biology*, vol. 6, no. 5, 2005.

[39] S. Aitken, Y. Chen, and J. Bard, "OBO Explorer: an editor for Open Biomedical Ontologies in OWL.," *Bioinformatics*, vol. 24, no. 3, pp. 443–444, 2008.

[40] "OboInOwl: Mapping OBO to OWL," Retrieved on 16 April 2012. `http://www.bioontology.org/wiki/index.php/OboInOwl:Main_Page`.

[41] M. Hucka, A. Finney, H. M. Sauro, H. Bolouri, J. C. Doyle, H. Kitano, the rest of the SBML Forum:, A. P. Arkin, B. J. Bornstein,

D. Bray, A. Cornish-Bowden, A. A. Cuellar, S. Dronov, E. D. Gilles, M. Ginkel, V. Gor, I. I. Goryanin, W. J. Hedley, T. C. Hodgman, J. H. Hofmeyr, P. J. Hunter, N. S. Juty, J. L. Kasberger, A. Kremling, U. Kummer, N. Le Novère, L. M. Loew, D. Lucio, P. Mendes, E. Minch, E. D. Mjolsness, Y. Nakayama, M. R. Nelson, P. F. Nielsen, T. Sakurada, J. C. Schaff, B. E. Shapiro, T. S. Shimizu, H. D. Spence, J. Stelling, K. Takahashi, M. Tomita, J. Wagner, and J. Wang, "The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models," *Bioinformatics*, vol. 19, no. 4, pp. 524–531, 2003.

[42] C. M. Lloyd, M. D. Halstead, and P. F. Nielsen, "CellML: its future, present and past.," *Progress in biophysics and molecular biology*, vol. 85, no. 2-3, pp. 433–450, 2004.

[43] E. Demir, M. P. Cary, S. Paley, K. Fukuda, C. Lemer, I. Vastrik, G. Wu, P. D'Eustachio, C. Schaefer, J. Luciano, F. Schacherer, I. Martinez-Flores, Z. Hu, V. Jimenez-Jacinto, G. Joshi-Tope, K. Kandasamy, A. C. Lopez-Fuentes, H. Mi, E. Pichler, I. Rodchenkov, A. Splendiani, S. Tkachev, J. Zucker, G. Gopinath, H. Rajasimha, R. Ramakrishnan, I. Shah, M. Syed, N. Anwar, O. Babur, M. Blinov, E. Brauner, D. Corwin, S. Donaldson, F. Gibbons, R. Goldberg, P. Hornbeck, A. Luna, P. Murray-Rust, E. Neumann, O. Reubenacker, M. Samwald, M. van Iersel, S. Wimalaratne, K. Allen, B. Braun, M. Whirl-Carrillo, K.-H. Cheung, K. Dahlquist, A. Finney, M. Gillespie, E. Glass, L. Gong, R. Haw, M. Honig, O. Hubaut, D. Kane, S. Krupa, M. Kutmon, J. Leonard, D. Marks, D. Merberg, V. Petri, A. Pico, D. Ravenscroft, L. Ren, N. Shah, M. Sunshine, R. Tang, R. Whaley, S. Letovksy, K. H. Buetow, A. Rzhetsky, V. Schachter, B. S. Sobral, U. Dogrusoz, S. McWeeney, M. Aladjem, E. Birney, J. Collado-Vides, S. Goto, M. Hucka, N. Le Novere, N. Maltsev, A. Pandey, P. Thomas, E. Wingender, P. D. Karp, C. Sander, and G. D. Bader, "The BioPAX community standard for pathway data sharing," *Nature Biotechnology*, vol. 28, no. 12, p. 1308, 2010.

[44] N. F. Noy, N. H. Shah, P. L. Whetzel, B. Dai, M. Dorf, N. Griffith, C. Jonquet, D. L. Rubin, M.-A. A. Storey, C. G. Chute, and M. A. Musen, "BioPortal: ontologies and integrated data resources at the

click of a mouse.," *Nucleic acids research*, vol. 37, no. Web Server issue, pp. W170–W173, 2009.

[45] R. Côté, F. Reisinger, L. Martens, H. Barsnes, J. A. A. Vizcaino, and H. Hermjakob, "The Ontology Lookup Service: bigger and better.," *Nucleic acids research*, vol. 38, no. Web Server issue, pp. W155–160, 2010.

[46] Z. Xiang, C. Mungall, A. Ruttenberg, and Y. He, "Ontobee: A Linked Data Server and Browser for Ontology Terms.," *Proc. of the International Conference on Biomedical Ontologies (ICBO)*, pp. 279–281, 2011.

[47] F. Belleau, M. Nolin, N. Tourigny, P. Rigault, and J. Morissette, "Bio2RDF: Towards a mashup to build bioinformatics knowledge systems," *Journal of Biomedical Informatics*, vol. 41, no. 5, pp. 706–716, 2008.

[48] "Google," Retrieved on 16 April 2012. `http://www.google.com/`.

[49] T. Judd and G. Kennedy, "Expediency-based practice? Medical students' reliance on Google and Wikipedia for biomedical inquiries," *British Journal of Educational Technology*, vol. 42, no. 2, pp. 351–360, 2011.

[50] "Wikipedia, The Free Encyclopedia," Retrieved on 16 April 2012. `http://www.wikipedia.org/`.

[51] M. Krötzsch, D. Vrandecic, and M. Völkel, "Wikipedia and the Semantic Web - The Missing Links," in *Proceedings of Wikimania - The First International Wikimedia Conference*, Wikimedia Foundation, 2005.

[52] E. Hodis, J. Prilusky, E. Martz, I. Silman, J. Moult, and J. Sussman, "Proteopedia - a scientific 'wiki' bridging the rift between three-dimensional structure and function of biomacromolecules," *Genome Biology*, vol. 9, no. 8, pp. R121+, 2008.

[53] A. E. Jinha, "Article 50 million: an estimate of the number of scholarly articles in existence," *Learned Publishing*, vol. 23, no. 3, pp. 258–263, 2010.

[54] "PubMed," Retrieved on 16 April 2012. `http://www.ncbi.nlm.nih.gov/pubmed/`.

[55] K. Bahaadinbeigy, K. Yogesan, and R. Wootton, "MEDLINE versus EMBASE and cumulative index to nursing and allied health literature for telemedicine searches.," *Telemedicine journal and e-health: the official journal of the American Telemedicine Association*, 2010.

[56] R. Feldman and J. Sanger, *The Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data*. Cambridge University Press, 2006.

[57] S. Van Landeghem, B. De Baets, Y. Van de Peer, and Y. Saeys, "High-precision bio-molecular event extraction from text using parallel binary classifiers," *Computational Intelligence*, vol. 27, no. 4, pp. 645–664, 2011.

[58] I. Spasic, S. Ananiadou, J. McNaught, and A. Kumar, "Text mining and ontologies in biomedicine: making sense of raw text," *Briefings in bioinformatics*, vol. 6, no. 3, pp. 239–251, 2005.

[59] A. Eaton, "HubMed: a web-based biomedical literature search interface," *Nucleic Acids Research*, vol. 34, no. suppl 2, pp. W745–W747, 2006.

[60] J. M. Fernández, R. Hoffmann, and A. Valencia, "iHOP web services.," *Nucleic acids research*, vol. 35, no. Web Server issue, 2007.

[61] H. Chen and B. M. Sharp, "Content-rich biological network constructed by mining PubMed abstracts.," *BMC Bioinformatics*, vol. 5, no. 1, pp. 147+, 2004.

[62] "BioMinT," Retrieved on 16 April 2012. `http://biomint.org`.

[63] S. Vercruysse and M. Kuiper, "Simulating genetic networks made easy: network construction with simple building blocks," *Bioinformatics*, vol. 21, no. 2, pp. 269+, 2005.

[64] D. Charles, "Aristotle on meaning and essence," *The Review of Metaphysics*, vol. 1, no. 9, pp. 1–410, 2000.

[65] G. Boole, *The Mathematical Analysis of Logic*. 1847.

[66] G. Frege, *Begriffsschrift: eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*. 1879.

[67] J. van Heijenoort, *From Frege To Gödel: A Source Book in Mathematical Logic, 1879-1931*. 1977.

[68] A. Fraenkel, Y. Bar-Hillel, and A. Levy, *Foundations of Set Theory.* 1973.

[69] E. Nagel and J. R. Newman, *Gödel's Proof.* New York University Press, 1958.

[70] A. Church, "A note on the entscheidungsproblem," *Journal of Symbolic Logic*, no. 1, pp. 40–41, 1936.

[71] A. Turing, "On computable numbers, with an application to the entscheidungsproblem," *Proceedings of the London Mathematical Society*, no. 2, p. 230–265, 1937.

[72] A. Turing, "Intelligent machinery," *report for the National Physical Laboratory*, 1948.

[73] L. Fortnow and S. Homer, "A short history of computational complexity," in *The History of Mathematical Logic*, 2002.

[74] S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the third annual ACM Symposium on Theory of Computing*, STOC '71, (New York, USA), pp. 151–158, ACM Press, 1971.

[75] B. Russell, "The philosophy of logical atomism," *The Collected Papers of Bertrand Russell*, no. 8, p. 228, 1914.

[76] I. Bratko, *Prolog Programming for Artificial Intelligence*. Addison Wesley, 3rd ed., 2000.

[77] A. K. Chandra and D. Harel, "Horn clause queries and generalizations," *The Journal of Logic Programming*, vol. 2, no. 1, pp. 1–15, 1985.

[78] R. J. Brachman and J. G. Schmolze, "An overview of the KL-ONE knowledge representation system," *Cognitive Science: A Multidisciplinary Journal*, vol. 9, no. 2, pp. 171–216, 1985.

[79] R. Weida, E. Mays, R. Dionne, M. Laker, B. White, C. Liang, and F. J. Oles, "The K-Rep System Architecture," in *the Proceedings of the 1996 International Workshop on Description Logics*, 1996.

[80] C. Peltason, "The BACK system - an overview," *SIGART Bull.*, vol. 2, pp. 114–119, 1991.

[81] R. M. MacGregor and R. Bates, "The Loom knowledge representation language," Tech. Rep. ISI/RS-87-188, Information Science Institute, University of Southern California, 1987.

[82] M. Schmidt-Schaubß, "Subsumption in KL-ONE is undecidable," in *Proceedings of the first international conference on Principles of knowledge representation and reasoning*, (San Francisco, CA, USA), pp. 421–431, Morgan Kaufmann Publishers Inc., 1989.

[83] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, eds., *The Description Logic Handbook: Theory, Implementation, and Applications*, Cambridge University Press, 2003.

[84] M. Schmidt-Schauß and G. Smolka, "Attributive concept descriptions with complements," *Artif. Intell.*, vol. 48, no. 1, pp. 1–26, 1991.

[85] F. Baader, I. Horrocks, and U. Sattler, "Description Logics," in *Handbook of Knowledge Representation* (F. van Harmelen, V. Lifschitz, and B. Porter, eds.), ch. 3, pp. 135–180, Elsevier, 2008.

[86] B. Thalheim, *Entity-Relationship Modeling: Foundations of Database Technology*. Springer, 1 ed., 2000.

[87] A. Artale, D. Calvanese, R. Kontchakov, V. Ryzhikov, and M. Zakharyaschev, "Complexity of Reasoning in Entity Relationship Models," in *Proceedings of the International Workshop on Description Logics (DL2007), Brixen-Bressanone*, vol. 250, 2007.

[88] T. Irwin, "The Science of Being," *Aristotle's First Principles*, vol. 1, no. 9, pp. 179–199, 1990.

[89] R. Gockel, *Lexicon philosophicum, quo tantam clave philosophiae fores aperiuntur*. (reprint: Hildesheim: Georg Olms, 1980), 1613.

[90] T. R. Gruber, "A translation approach to portable ontology specifications," *Knowl. Acquis.*, vol. 5, pp. 199–220, 1993.

[91] W. Ceusters and B. Smith, "A realism-based approach to the evolution of biomedical ontologies," *AMIA Annual Symposium proceedings / AMIA Symposium*, pp. 121–125, 2006.

[92] D. M. Armstrong, *Universals: an opinionated introduction*. Westview Press, Boulder, 1989.

[93] M. Woods, "Universals and Particular Forms in Aristotle's Metaphysics," *Oxford Studies in Ancient Philosophy*, vol. Aristotle and the Later Tradition, pp. 41–56, 1991.

[94] "HTML 4 – Conformance: requirements and recommendations," Retrieved on 16 April 2012. `http://www.w3.org/TR/html401/conform.html`.

[95] C. Musciano and B. Kennedy, *HTML & XHTML : The Definitive Guide*. O'Reilly, 4 ed., 2000.

[96] R. T. Fielding, J. Gettys, J. C. Mogul, L. Masinter, P. J. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol — HTTP/1.1," Tech. Rep. 2616, RFC Editor, Fremont, CA, USA, 1999.

[97] T. Berners-Lee, "Information Management: A Proposal," tech. rep., CERN, 1989.

[98] "World Wide Web Consortium (W3C)," Retrieved on 16 April 2012. `http://www.w3.org/`.

[99] "The Semantic Web," Retrieved on 16 April 2012. `http://www.w3.org/standards/semanticweb/`.

[100] I. Horrocks, B. Parsia, P. P. Schneider, and J. Hendler, *Semantic Web Architecture: Stack or Two Towers?*, pp. 37–41. No. 3703 in LNCS, SV, 2005.

[101] "About the Unicode Standard," Retrieved on 16 April 2012. `http://unicode.org/standard/standard.html`.

[102] M. Duerst and M. Suignard, "Internationalized Resource Identifiers (IRIs)." RFC 3987, 2005.

[103] T. Bray, J. Paoli, C. M. Sperberg-Mcqueen, Eve, and F. Yergeau, eds., *Extensible Markup Language (XML) 1.0*. W3C Recommendation, W3C, fourth ed., 2003, Retrieved on 16 April 2012. `http://www.w3.org/TR/REC-xml/`.

[104] E. Miller and F. Manola, "RDF Primer," W3C Recommendation, W3C, 2004, Retrieved on 16 April 2012. `http://www.w3.org/TR/2004/REC-rdf-primer-20040210/`.

[105] J. J. Carroll and G. Klyne, "Resource description framework (RDF): Concepts and abstract syntax," W3C recommendation, W3C, 2004, Retrieved on 16 April 2012. `http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/`.

[106] E. Prud'hommeaux and A. Seaborne, "SPARQL Query Language for RDF." W3C Recommendation, 2008, Retrieved on 16 April 2012. `http://www.w3.org/TR/rdf-sparql-query/`.

[107] R. V. Guha and D. Brickley, "RDF Vocabulary Description Language 1.0: RDF Schema," W3C Recommendation, W3C, 2004, Retrieved on 16 April 2012. `http://www.w3.org/TR/2004/REC-rdf-schema-20040210/`.

[108] G. Schreiber and M. Dean, "OWL Web Ontology Language Reference," W3C Recommendation, W3C, 2004, Retrieved on 16 April 2012. `http://www.w3.org/TR/2004/REC-owl-ref-20040210/`.

[109] D. L. McGuinness and F. van Harmelen, "OWL web ontology language overview," W3C recommendation, W3C, 2004, Retrieved on 16 April 2012. `http://www.w3.org/TR/2004/REC-owl-features-20040210/`.

[110] B. Motik, A. Fokoue, I. Horrocks, Z. Wu, C. Lutz, and B. C. Grau, "OWL 2 Web Ontology Language Profiles," W3C recommendation, W3C, 2009, Retrieved on 16 April 2012. `http://www.w3.org/TR/2009/REC-owl2-profiles-20091027/`.

[111] J. D. Becker, "Unicode 88," tech. rep., Xerox Corp., Palo Alto, CA, USA, 1988.

[112] F. Yergeau, "UTF-8, a transformation format of ISO 10646," tech. rep., 2003, Retrieved on 16 April 2012. `http://datatracker.ietf.org/doc/rfc3629/`.

[113] American National Standard for Information Systems, "Coded Character Sets – 7-Bit American National Standard Code for Information Interchange (7-Bit ASCII), ANSI X3.4-1986," *American National Standards Institute, Inc.*, 1986.

[114] T. Berners-Lee, R. T. Fielding, and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax," RFC 3986, RFC Editor, Fremont, CA, USA, 2005.

[115] T. Berners-Lee, "Uniform Resource Locators," 1994, Retrieved on 16 April 2012. `http://www.w3.org/Addressing/URL/url-spec.txt`.

[116] M. Feeney, *The Standard Generalized Markup Language (SGML)*. London, UK: British Library Research and Development Dept. and Library & Information Technology Centre, 1988.

[117] C. F. Goldfarb and Y. Rubinsky, *The SGML Handbook*. Oxford: Clarendon Press, 1990.

[118] D. Olteanu, H. Meuss, T. Furche, and F. Bry, "XPath: Looking Forward," *XML-Based Data Management and Multimedia Engineering - EDBT 2002 Workshops*, pp. 892–896, 2002.

[119] D. Chamberlin, "XQuery: a query language for XML," in *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, SIGMOD '03, (New York, NY, USA), p. 682, ACM, 2003.

[120] T. Berners-Lee, "The WorldWideWeb browser," 1990, Retrieved on 16 April 2012. `http://www.w3.org/People/Berners-Lee/WorldWideWeb.html`.

[121] D. Raggett, A. Le Hors, and I. Jacobs, "HTML 4.01 Specification," tech. rep., 1999, Retrieved on 16 April 2012. `http://www.w3.org/TR/html4/`.

[122] S. Pemberton, "XHTML 1.0: The extensible hypertext markup language - a reformulation of HTML 4 in XML 1.0," first Edition of a Recommendation, W3C, 2000, Retrieved on 16 April 2012. `http://www.w3.org/TR/2000/REC-xhtml1-20000126`.

[123] D. Beckett, "RDF/XML Syntax Specification," tech. rep., 2004, Retrieved on 16 April 2012. `www.w3.org/TR/REC-rdf-syntax/`.

[124] J. Melton, A. Simpson, and J. Gray, *Understanding the new SQL: A Complete Guide*. Morgan Kaufmann, 1993.

[125] J. Pérez, M. Arenas, and C. Gutierrez, "Semantics and Complexity of SPARQL," pp. 30–43, 2006.

[126] A. Seaborne, G. Manjunath, C. Bizer, J. Breslin, S. Das, I. Davis, S. Harris, K. Idehen, O. Corby, K. Kjernsmo, and B. Nowack, "SPARQL Update, A language for updating RDF graphs," W3C Member Submission, 2008, Retrieved on 16 April 2012. `http://www.w3.org/Submission/SPARQL-Update/`.

[127] D. Beckett, T. Berners-Lee, and E. Prud'hommeaux, "Turtle, Terse RDF Triple Language," W3C Team Submission, 2011, Retrieved on 16 April 2012. `http://www.w3.org/TeamSubmission/turtle/`.

[128] T. Berners-Lee and D. Connolly, "Notation3 (N3): A readable RDF syntax," W3C Team Submission, W3C, 2011, Retrieved on 16 April 2012. `http://www.w3.org/TeamSubmission/n3/`.

[129] D. Fensel, F. van Harmelen, I. Horrocks, D. L. Mcguinness, and P. F. Patel-Schneider, "OIL: an ontology infrastructure for the semantic web," *Intelligent Systems, IEEE [see also IEEE Intelligent Systems and Their Applications]*, vol. 16, no. 2, pp. 38–45, 2001.

[130] D. Connolly, F. Harmelen, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein, "DAML+OIL: Reference description," W3C Note, 2001, Retrieved on 16 April 2012. `http://www.w3.org/TR/daml+oil-reference`.

[131] I. Horrocks and P. Patel-Schneider, "Reducing OWL entailment to description logic satisfiability," in *Semantic Web - ISWC 2003*

(Fensel, D and Sycara, K and Mylopoulos, J, ed.), vol. 2870 of *Lecture Notes in Computer Science*, pp. 17–29, 2003.

[132] R. Hoehndorf, M. Dumontier, A. Oellrich, S. Wimalaratne, D. Rebholz-Schuhmann, P. Schofield, and G. V. Gkoutos, "A common layer of interoperability for biomedical ontologies based on OWL EL," *Bioinformatics*, vol. 27, no. 7, pp. 1001–1008, 2011.

[133] P. F. Patel-Schneider, B. Motik, and B. C. Grau, "OWL 2 Web Ontology Language Direct Semantics," W3C recommendation, W3C, 2009, Retrieved on 16 April 2012. `http://www.w3.org/TR/2009/REC-owl2-direct-semantics-20091027/`.

[134] M. Schneider, "OWL 2 Web Ontology Language RDF-Based Semantics," W3C recommendation, W3C, 2009, Retrieved on 16 April 2012. `http://www.w3.org/TR/2009/REC-owl2-rdf-based-semantics-20091027/`.

[135] E. Sirin, B. Parsia, B. Grau, A. Kalyanpur, and Y. Katz, "Pellet: A practical OWL-DL reasoner," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 5, no. 2, pp. 51–53, 2007.

[136] U. Hustadt, B. Motik, and U. Sattler, "Data Complexity of Reasoning in Very Expressive Description Logics," in *Proceedings of the International Joint Conferences on Artificial Intelligence (IJCAI)*, pp. 466–471, Professional Book Center, 2005.

[137] E. Sirin and B. Parsia, "SPARQL-DL: SPARQL Query for OWL-DL," in *3rd OWL: Experiences and Directions Workshop (OWLED)*, 2007.

[138] I. Horrocks, P. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean, "SWRL: A Semantic Web Rule Language Combining OWL and RuleML," W3C Member Submission, W3C, 2004, Retrieved on 16 April 2012. `http://www.w3.org/Submission/SWRL/`.

[139] S. T. H. Boley and G. Wagner, "Design rationale of RuleML: A markup language for semantic web rules," in *Proc. Semantic Web Working Symposium* (I. F. Cruz, S. Decker, J. Euzenat, and D. L. McGuinness, eds.), (Stanford University, California), pp. 381–402, 2001.

[140] A. Paschke, D. Reynolds, G. Hallmark, H. Boley, M. Kifer, and A. Polleres, "RIF Core Dialect," W3C Recommendation, W3C, 2010, Retrieved on 16 April 2012. `http://www.w3.org/TR/rif-core/`.

[141] H. Knublauch, J. Hendler, and K. Idehen, "SPIN - Overview and Motivation," W3C Member Submission, W3C, 2011, Retrieved on 16 April 2012. `http://www.w3.org/Submission/spin-overview/`.

[142] T. Berners-Lee, "Linked Data," tech. rep., 2006, Retrieved on 16 April 2012. `http://www.w3.org/DesignIssues/LinkedData.html`.

[143] C. Bizer, T. Heath, and T. Berners-Lee, "Linked Data - The Story So Far," *International Journal on Semantic Web and Information Systems*, vol. 5, no. 3, pp. 1–22, 2009.

[144] E. Prud'hommeaux and C. Buil-Aranda, "SPARQL 1.1 Federated Query, W3C Working Draft 10 November 2011," 2011, Retrieved on 16 April 2012. `http://www.w3.org/2009/sparql/docs/fed/service`.

[145] A. Venkatesan, W. Blondé, E. Antezana, M. Skillingstad, M. S. Marshall, B. De Baets, V. Mironov, and M. Kuiper, "The RDF foundry: call for an initiative to build enhanced RDF resources for biological data integration," in *Proceedings of the International Conference on Web Intelligence, Mining and Semantics*, WIMS '11, (New York, NY, USA), pp. 59:1–59:5, ACM, 2011.

[146] M. Widenius, D. Axmark, and A. B. Mysql, *MySQL Reference Manual*. O'Reilly Media, Inc., 1 ed., 2002.

[147] D. J. Higham and N. J. Higham, *MATLAB Guide*. SIAM: Society for Industrial and Applied Mathematics, 2005.

[148] "The DOT Language," Retrieved on 16 April 2012. `http://www.graphviz.org/doc/info/lang.html`.

[149] E. Antezana, E. Tsiporkova, V. Mironov, and M. Kuiper, "A cell-cycle knowledge integration framework," in *Proceedings of the Third international conference on Data Integration in the Life Sciences*, DILS'06, (Berlin, Heidelberg), pp. 19–34, Springer-Verlag, 2006.

[150] "DIAMONDS project," Retrieved on 16 April 2012. `http://www.sbcellcycle.org/`.

[151] "Graphlet: The GML File Format," Retrieved on 16 April 2012. `http://www.webcitation.org/query.php?url=http://www.infosun.fim.uni-passau.de/Graphlet/GML/&refdoi=10.1186/gb-2009-10-5-r58`.

[152] M. Aranguren, E. Antezana, M. Kuiper, and R. Stevens, "Ontology Design Patterns for bio-ontologies: a case study on the Cell Cycle Ontology," *BMC Bioinformatics*, vol. 9, no. Suppl 5, pp. S1+, 2008.

[153] O. Erling and I. Mikhailov, "RDF Support in the Virtuoso DBMS Networked Knowledge - Networked Media," vol. 221 of *Studies in Computational Intelligence*, ch. 2, pp. 7–24, Berlin, Heidelberg: Springer, 2009.

[154] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of Molecular Biology*, vol. 215, no. 3, pp. 403–410, 1990.

[155] C. Bizer and A. Schultz, "The Berlin SPARQL Benchmark," *International Journal on Semantic Web and Information Systems*, vol. 5, no. 2, pp. 1–24, 2009.

[156] Y. Guo, Z. Pan, and J. Heflin, "LUBM: A benchmark for OWL knowledge base systems," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 3, no. 2-3, pp. 158–182, 2005.

[157] M. Morsey, J. Lehmann, S. Auer, and A.-C. N. Ngomo, "DBpedia SPARQL benchmark: performance assessment with real queries on real data," in *Proceedings of the 10th international conference on The semantic web - Volume Part I*, ISWC'11, (Berlin, Heidelberg), pp. 454–469, Springer-Verlag, 2011.

[158] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives, "DBpedia: A Nucleus for a Web of Open Data The Semantic Web," in *Proceedings of 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference (ISWC+ASWC)*, vol. 4825 of *Lecture Notes in Computer Science*, ch. 52, pp. 722–735, Berlin, Heidelberg: Springer, 2007.

[159] V. Mironov, N. Seethappan, W. Blondé, E. Antezana, B. Lindi, and M. Kuiper, "Benchmarking triple stores with biological data," *CoRR*, vol. abs/1012.1632, 2010.

[160] J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson, "Jena: implementing the semantic web recommendations," in *WWW Alt. '04: Proceedings of the 13th international World Wide Web conference on Alternate track papers*, (New York, NY, USA), pp. 74–83, ACM, 2004.

[161] "4store - Scalable RDF storage," Retrieved on 16 April 2012. `http://4store.org/`.

[162] A. Kiryakov, D. Ognyanov, and D. Manov, "OWLIM – A Pragmatic Semantic Repository for OWL," *Web Information Systems Engineering – WISE 2005 Workshops*, pp. 182–192, 2005.

[163] A. Chebotko, S. Lu, M. Atay, and F. Fotouhi, "Efficient Processing of RDF Queries with Nested Optional Graph Patterns in an RDBMS," *International Journal on Semantic Web and Information Systems*, vol. 4, no. 4, pp. 1–30, 2008.

[164] F. Fischer, G. Unel, B. Bishop, and D. Fensel, "Towards a Scalable, Pragmatic Knowledge Representation Language for the Web," in *Perspectives of Systems Informatics. 7th International Andrei Ershov Memorial Conference, PSI 2009. Revised Papers*, pp. 231–41, 2010.

[165] X. Bai, R. Delbru, and G. Tummarello, "RDF Snippets for Semantic Web Search Engines," in *On the Move to Meaningful Internet Systems*, vol. 5332 of *Lecture Notes in Computer Science (LNCS)*, pp. 1304–1318, 2008.

[166] "Google Webmaster Central Blog: Introducing Rich Snippets," tech. rep., 2009, Retrieved on 16 April 2012. `http://googlewebmastercentral.blogspot.com/2009/05/introducing-rich-snippets.html`.

[167] "Owl web ontology language reference: Properties," 2004, Retrieved on 16 April 2012. `http://www.w3.org/TR/owl-ref/#Property`.

[168] J. F. Nilsson, "Querying Class-Relationship Logic in a Metalogic Framework," in *Flexible Query Answering Systems (FQAS)*, pp. 96–107, 2011.

[169] S. Zambach and J. U. Hansen, "Logical Knowledge Representation of Regulatory Relations in Biomedical Pathways," in *International Conference on Information Technology in Bio- and Medical Informatics (ITBAM)*, pp. 186–200, 2010.

[170] C. Darwin, *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*. J. Murray, sixth ed., 1872.

[171] H. Kitano, "Systems biology: A brief overview," *Science*, vol. 295, no. 5560, pp. 1662–1664, 2002.

[172] E. Antezana, W. Blondé, A. Venkatesan, B. De Baets, V. Mironov, and M. Kuiper, "Semantic systems biology: enabling integrative biology via semantic web technologies," in *Proceedings of the International Conference on Web Intelligence, Mining and Semantics*, WIMS '11, (New York, NY, USA), pp. 58:1–58:5, ACM, 2011.

[173] "Obo ontologies," Retrieved on 16 April 2012. `http://www.berkeleybop.org/ontologies/`.

[174] "OBO specification 1.2," Retrieved on 16 April 2012. `http://www.geneontology.org/GO.format.obo-1_2.shtml`.

[175] "ONTO-PERL distribution," Retrieved on 16 April 2012. `http://search.cpan.org/dist/ONTO-PERL/`.

[176] "Spatial Ontology," Retrieved on 16 April 2012. `http://obofoundry.org/cgi-bin/detail.cgi?id=spatial`.

[177] "Worm Anatomy Ontology," Retrieved on 16 April 2012. `http://www.obofoundry.org/cgi-bin/detail.cgi?id=worm_anatomy`.

[178] "Adobe Flex," Retrieved on 16 April 2012. `http://www.adobe.com/products/flex/`.

[179] "Netthreads," Retrieved on 16 April 2012. `http://www.netthreads.co.uk/`.

[180] "Python query interface to BioGateway SPARQL endpoint and InterMine," Retrieved on 16 April 2012. `http://bcbio.wordpress.com/2010/02/15/python-query-interface-to-biogateway-sparql-endpoint-and-intermine/`.

[181] S. Koehler, S. Bauer, C. J. Mungall, G. Carletti, C. L. Smith, P. Schofield, G. V. Gkoutos, and P. N. Robinson, "Improving ontologies by automatic reasoning and evaluation of logical definitions," *BMC Bioinformatics*, vol. 12, 2011.

[182] B. M. Good and M. D. Wilkinson, "The Life Sciences Semantic Web is full of creeps!," *Briefings in Bioinformatics*, vol. 7, no. 3, pp. 275–286, 2006.

[183] E. Antezana, *Towards semantic systems biology : biological knowledge management using semantic web technologies*. PhD thesis, Ghent University, 2009.

Appendix A

Metarel

Metarel was first developed in the OBO format, but was later translated into the XML syntax of RDF and maintained in that format. This has made it somewhat less human readable, however, XML provides some flexibility for creating representations that are easy to read and understand. Since the RDF representation of Metarel is maintained manually and validated with RDF parsers, a somewhat readable code could be preserved.

The most important features for each term in Metarel are the following:

- The IRI, tagged with *rdf:Description*.
- The label, tagged with *rdfs:label*.
- The definition, tagged with *rdfs:comment*.
- The more generic term, tagged with *rdfs:subClassOf*.

In this manner, Metarel can be read and 'understood' by both human and computers. The RDF code can be loaded in every RDF store, where it can be merged with relation types in the RDF representation.

metarel.rdf:

```
<?xml version="1.0"?>
<rdf:RDF
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
        xmlns:owl="http://www.w3.org/2002/07/owl#"
        xmlns:metarel="http://www.metarel.org/">
        <rdf:Description rdf:about="http://www.metarel.org/AllAllClassRelationType">
                <rdfs:label>all-all class relation type</rdfs:label>
                <rdfs:comment>An all-all class relation type is a class relation
```

```
        type for which all the class relation arcs are total. This kind
        of class relation types may be useful to construct from instance
        relation types that express a disposition. E.g all the proteins of
        protein class A have the disposition to interact with any protein
        of protein class B.</rdfs:comment>
        <rdfs:subClassOf rdf:resource=
        "http://www.metarel.org/InvertibleInstanceBasedClassRelationType"/>
</rdf:Description>
<rdf:Description rdf:about="http://www.metarel.org/AllSomeClassRelationType">
        <rdfs:label>all-some class relation type</rdfs:label>
        <rdfs:comment>An all-some class relation type is an instance-based
        class relation type of which all the relation arcs should be total
        and are allowed to be not surjective. A class relation arc is total
        if all the instances of the source class have at least one
        corresponding instance relation arc to the target class. It is
        surjective if all the instances from the target class receive at
        least one corresponding instance relation arc from the source class.
        E.g. 'tusk is part of animal' is a total arc as every tusk is part
        of an animal. 'Animal has part tusk' is not total and therefore not
        allowed for an all-some relation type.</rdfs:comment>
        <rdfs:subClassOf
        rdf:resource="http://www.metarel.org/ClassRelationType"/>
        <rdfs:subClassOf
        rdf:resource="http://www.metarel.org/InstanceBasedRelationType"/>
</rdf:Description>
<rdf:Description rdf:about="http://www.metarel.org/ClassDataRelationType">
        <rdfs:label>class-data relation type</rdfs:label>
        <rdfs:comment>A class-data relation type is a relation type of
        which all the relations are from a class to a data type.
        </rdfs:comment>
        <rdfs:subClassOf
        rdf:resource="http://www.metarel.org/SubjectDataRelationType"/>
</rdf:Description>
<rdf:Description rdf:about="http://www.metarel.org/ClassRelationType">
        <rdfs:label>class relation type</rdfs:label>
        <rdfs:comment>Any relation type of which all the relations are
        between two classes is a class relation type.</rdfs:comment>
        <rdfs:subClassOf
        rdf:resource="http://www.metarel.org/SubjectObjectRelationType"/>
</rdf:Description>
<rdf:Description rdf:about="http://www.metarel.org/CompositionalClosure">
        <rdfs:label>compositional closure</rdfs:label>
        <rdfs:comment>A compositional closure is an inference mechanism
        that infers the resulting relation types of relational composites.
        </rdfs:comment>
        <rdfs:subClassOf
        rdf:resource="http://www.metarel.org/RelationalClosure"/>
</rdf:Description>
<rdf:Description rdf:about="http://www.metarel.org/DefiningRelationType">
        <rdfs:label>defining relation type</rdfs:label>
        <rdfs:comment>A defining relation type is a relation type whose
        relation arcs are part of a set of necessary and sufficient
        conditions for instances to be an instance of a subject class.
        All the defining relation types arcs that point away from a
        certain class, form a set of necessary and sufficient conditions
        together. This set defines that class.</rdfs:comment>
        <rdfs:subClassOf
        rdf:resource="http://www.metarel.org/SubjectObjectRelationType"/>
</rdf:Description>
<rdf:Description rdf:about="http://www.metarel.org/FixedComplexComposite">
        <rdfs:label>fixed complex composite</rdfs:label>
        <rdfs:comment>The first, the second and the resulting relation
        type are all different.\nIf the first and the second relation
        type change their positions, then it\ndoes not give the same
        resulting relation type any longer. E.g. 'is brother of' +
        'is father of' results in 'is uncle of'. In DL notation
        %3A R o S %3C T.</rdfs:comment>
        <rdfs:subClassOf
        rdf:resource="http://www.metarel.org/RelationalComposite"/>
</rdf:Description>
<rdf:Description rdf:about="http://www.metarel.org/FixedPriorityComposite">
        <rdfs:label>fixed priority composite</rdfs:label>
        <rdfs:comment>The resulting relation type is the same as the
        second relation type, but different from the first relation type.
```

```
        If the first and the second relation type change their positions,
        then it does not give the same resulting relation type any
        longer. E.g. 'is sibling of' + 'is child of' results in 'is child
        of'. In DL notation%3A R o S %3C S.</rdfs:comment>
        <rdfs:subClassOf
        rdf:resource="http://www.metarel.org/RelationalComposite"/>
</rdf:Description>
<rdf:Description rdf:about="http://www.metarel.org/InferenceMechanism">
        <rdfs:label>inference mechanism</rdfs:label>
        <rdfs:comment>An inference mechanism describes a systematic
        procedure on how relation arcs can be inferred in a Knowledge
        Base. This procedure can be formalized with construct queries.
        </rdfs:comment>
        <rdfs:subClassOf
        rdf:resource="http://www.metarel.org/SemanticsProvidingConstruct"/>
</rdf:Description>
<rdf:Description rdf:about="http://www.metarel.org/InstanceBasedRelationType">
        <rdfs:label>instance-based relation type</rdfs:label>
        <rdfs:comment>An instance-based relation type is a relation type
        that is based on an instance relation type by using logical
        quantifiers. E.g. if all the instances of the class 'leg' are
        part of some instances of the class 'body', we can derive the
        instance-based relation arc 'leg is part of body'.</rdfs:comment>
        <rdfs:subClassOf
        rdf:resource="http://www.metarel.org/SubjectObjectRelationType"/>
</rdf:Description>
<rdf:Description rdf:about="http://www.metarel.org/InstanceClassRelationType">
        <rdfs:label>instance-class relation type</rdfs:label>
        <rdfs:comment>Any relation type of which all the relations are
        from an instance to a class is an instance-class relation type.
        </rdfs:comment>
        <rdfs:subClassOf
        rdf:resource="http://www.metarel.org/SubjectObjectRelationType"/>
</rdf:Description>
<rdf:Description rdf:about="http://www.metarel.org/InstanceDataRelationType">
        <rdfs:label>instance-data relation type</rdfs:label>
        <rdfs:comment>An instance-data relation type is a relation type
        of which all the relations are from an instance to a data type.
        </rdfs:comment>
        <rdfs:subClassOf
        rdf:resource="http://www.metarel.org/SubjectDataRelationType"/>
</rdf:Description>
<rdf:Description rdf:about="http://www.metarel.org/InstanceRelationType">
        <rdfs:label>instance relation type</rdfs:label>
        <rdfs:comment>Any relation type of which all the relations are
        between two instances is an instance relation type.
        </rdfs:comment>
        <rdfs:subClassOf
        rdf:resource="http://www.metarel.org/SubjectObjectRelationType"/>
</rdf:Description>
<rdf:Description
rdf:about="http://www.metarel.org/InterchangeableComplexComposite">
        <rdfs:label>interchangeable complex composite</rdfs:label>
        <rdfs:comment>The first, the second and the resulting relation
        type are all different. If the first and the second relation type
        change their positions, it still gives the same resulting relation
        type. E.g. 'is father of' + 'is grandfather of' results in 'is
        great-grandfather of'. In DL notation%3A R o S %3C T or S o R %3C
        T.</rdfs:comment>
        <rdfs:subClassOf
        rdf:resource="http://www.metarel.org/RelationalComposite"/>
</rdf:Description>
<rdf:Description
rdf:about="http://www.metarel.org/InterchangeablePriorityComposite">
        <rdfs:label>interchangeable priority composite</rdfs:label>
        <rdfs:comment>The resulting relation type is the same as the
        first or the second relation type, but the first and the second
        are different. If the first and the second relation type change
        their positions, it still gives the same resulting relation type.
        E.g. 'is part of' + 'is a' results in 'is part of' In DL notation
        %3A R o S %3C R or S o R %3C R.</rdfs:comment>
        <rdfs:subClassOf
        rdf:resource="http://www.metarel.org/RelationalComposite"/>
</rdf:Description>
```

```
<rdf:Description
rdf:about="http://www.metarel.org/InvertibleInstanceBasedClassRelationType">
        <rdfs:label>invertible instance-based class relation type
        </rdfs:label>
        <rdfs:comment>An invertible instance-based class relation type is
        an class relation type for which an inverse relation type exists
        that uses the same quantification method to base on an instance
        relation type. An all-some relation type is not invertible
        because it never has an inverse that is also an all-some relation
        type.</rdfs:comment>
        <rdfs:subClassOf
        rdf:resource="http://www.metarel.org/ClassRelationType"/>
        <rdfs:subClassOf
        rdf:resource="http://www.metarel.org/InstanceBasedRelationType"/>
</rdf:Description>
<rdf:Property rdf:about="http://www.metarel.org/isClassInstanceRelatedTo">
        <rdfs:label>is class-instance related to</rdfs:label>
        <rdfs:comment>A relation from an class to an instance is an 'is
        class-instance related to', if the relation is an instance of any
        class-instance relation type.</rdfs:comment>
        <owl:disjointWith
        rdf:resource="http://www.metarel.org/isClassRelatedTo"/>
        <owl:disjointWith
        rdf:resource="http://www.metarel.org/isInstanceClassRelatedTo"/>
        <owl:disjointWith
        rdf:resource="http://www.metarel.org/isInstanceRelatedTo"/>
        <rdfs:subClassOf rdf:resource="http://www.metarel.org/isRelatedTo"/>
</rdf:Property>
<rdf:Property rdf:about="http://www.metarel.org/isClassRelatedTo">
        <rdfs:label>is class related to</rdfs:label>
        <rdfs:comment>A relation between two classes is an 'is class
        related to', if the relation is an instance of any class relation
        type.</rdfs:comment>
        <owl:disjointWith
        rdf:resource="http://www.metarel.org/isClassInstanceRelatedTo"/>
        <owl:disjointWith
        rdf:resource="http://www.metarel.org/isInstanceClassRelatedTo"/>
        <owl:disjointWith
        rdf:resource="http://www.metarel.org/isInstanceRelatedTo"/>
        <rdfs:subClassOf rdf:resource="http://www.metarel.org/isRelatedTo"/>
</rdf:Property>
<rdf:Property rdf:about="http://www.metarel.org/isInstanceClassRelatedTo">
        <rdfs:label>is instance-class related to</rdfs:label>
        <rdfs:comment>A relation from an instance to a class is an 'is
        instance-class related to', if the relation is an instance of any
        instance-class relation type.</rdfs:comment>
        <owl:disjointWith
        rdf:resource="http://www.metarel.org/isClassInstanceRelatedTo"/>
        <owl:disjointWith
        rdf:resource="http://www.metarel.org/isClassRelatedTo"/>
        <owl:disjointWith
        rdf:resource="http://www.metarel.org/isInstanceRelatedTo"/>
        <rdfs:subClassOf rdf:resource="http://www.metarel.org/isRelatedTo"/>
</rdf:Property>
<rdf:Property rdf:about="http://www.metarel.org/isInstanceRelatedTo">
        <rdfs:label>is instance related to</rdfs:label>
        <rdfs:comment>A relation between two instances is an 'is instance
        related to', if the relation is an instance of any instance
        relation type.</rdfs:comment>
        <owl:disjointWith
        rdf:resource="http://www.metarel.org/isClassInstanceRelatedTo"/>
        <owl:disjointWith
        rdf:resource="http://www.metarel.org/isClassRelatedTo"/>
        <owl:disjointWith
        rdf:resource="http://www.metarel.org/isInstanceClassRelatedTo"/>
        <rdfs:subClassOf
        rdf:resource="http://www.metarel.org/isRelatedTo"/>
</rdf:Property>
<rdf:Property rdf:about="http://www.metarel.org/isRelatedTo">
        <rdfs:label>is related to</rdfs:label>
        <rdfs:comment>'Is related to' is the root term in the relation
        classification. This classification corresponds with a role
        hierarchy in Description Logics. A relation is the collection of
        all the relation arcs that start in the same source-term and end
```

```
        in the same target-term. Hence, between two terms, there can be
        at most two relations, one per direction.\nA class of relations
        is a relation type. 'Is related to' is the name of such a class.
        A relation can be an instance of different relation types. Every
        instantiation of a relation type by the relation corresponds with
        a relation arc.\nThe relation arcs are the asserted and
        visualised elements in ontologies. They are arcs that bear a
        label with the name of their corresponding relation type. In good
        ontologies often only one relation arc, or none at all, are
        asserted per relation. 'Is related to' would not be asserted in a
        good ontology, although much relations can contain a non-asserted
        arc with the name of the relation type 'is related to'.
        </rdfs:comment>
</rdf:Property>
<rdf:Description rdf:about="http://www.metarel.org/ReflexiveClosure">
        <rdfs:label>reflexive closure</rdfs:label>
        <rdfs:comment>A reflexive closure is an inference mechanism that
        infers all the self-arcs from reflexive relation types.
        </rdfs:comment>
        <rdfs:subClassOf
        rdf:resource="http://www.metarel.org/RelationalClosure"/>
</rdf:Description>
<rdf:Description rdf:about="http://www.metarel.org/ReflexiveRelationType">
        <rdfs:label>reflexive relation type</rdfs:label>
        <rdfs:comment>A relation type R is a reflexive relation type
        when A R A applies for all A in the domain of R.</rdfs:comment>
        <rdfs:subClassOf
        rdf:resource="http://www.metarel.org/SubjectObjectRelationType"/>
</rdf:Description>
<rdf:Description rdf:about="http://www.metarel.org/RelationalClosure">
        <rdfs:label>relational closure</rdfs:label>
        <rdfs:comment>A relational closure is a semantics-providing
        construct that provides meaning by describing an inference
        mechanism that has to be applied, often recursively, until new
        inferences cannot be made anymore. The order of the application
        of an inference mechanism should not matter among different
        relational closures.</rdfs:comment>
        <rdfs:subClassOf
        rdf:resource="http://www.metarel.org/InferenceMechanism"/>
</rdf:Description>
<rdf:Description rdf:about="http://www.metarel.org/RelationalComposite">
        <rdfs:label>relational composite</rdfs:label>
        <rdfs:comment>Three relation types r1, r2 and r3 (not
        necessarily all different) form a relational composite r1_r2_r3
        if for all the triples of nodes a, b, c (all different) for
        which there is an arc from a to b with the name of r1 and an arc
        from b to c with the name of r2, an arc from a to c with the
        name of r3 can be inferred. A relational composite is used to
        build role inclusion axioms (RIA). E.g. the first relation 'is
        brother of' and the second relation 'is father of' result in 'is
        uncle of'.</rdfs:comment>
        <rdfs:subClassOf
        rdf:resource="http://www.metarel.org/SemanticsProvidingConstruct"/>
</rdf:Description>
<rdf:Description
rdf:about="http://www.metarel.org/RelationalSubsumptionClosure">
        <rdfs:label>relational subsumption closure</rdfs:label>
        <rdfs:comment>A relational subsumption closure is an inference
        mechanism that infers an arc s from an arc r, where s has the
        label of the relation type S, r has the label of the relation
        type R, and R is a subrelation of S.</rdfs:comment>
        <rdfs:subClassOf
        rdf:resource="http://www.metarel.org/RelationalClosure"/>
</rdf:Description>
<rdf:Description
rdf:about="http://www.metarel.org/RelationOverSubsumptionClosure">
        <rdfs:label>relation-over-subsumption closure</rdfs:label>
        <rdfs:comment>A relation-over-subsumption closure is an
        inference mechanism that infers new relation arcs for all the
        all-some relation types, as these have priority over the
        subsumption relation%3A every all-some relation type forms an
        interchangeable priority composite with the subsumption
        relation. The relation-over-subsumption closure implies that all
        the relation arcs that follow from this mechanism, are inferred.
```

```
        </rdfs:comment>
        <rdfs:subClassOf
        rdf:resource="http://www.metarel.org/CompositionalClosure"/>
</rdf:Description>
<rdf:Description rdf:about="http://www.metarel.org/RelationType">
        <rdfs:label>relation type</rdfs:label>
        <rdfs:comment>A relation type is a class of relations. Such a
        class can have properties like transitivity, symmetry,
        reflexivity, invertibility, compositionality with other relation
        types, and more. The relation types are instances in the
        relation type classification, and are classified on such
        properties. Hence, the class named 'relation type' is a
        meta-class, being a class of classes. All the subclasses of
        'relation type' are also meta-classes.</rdfs:comment>
        <owl:sameAs
        rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
</rdf:Description>
<rdf:Description
rdf:about="http://www.metarel.org/RelevantRelationalSubsumptionClosure">
        <rdfs:label>relevant relational subsumption closure</rdfs:label>
        <rdfs:comment>A relevant relational subsumption closure is a
        relational subsumption closure where only the relation arcs are
        inferred that are considered as relevant for queries to the
        Knowledge Base. arcs with generic labels like 'is related to'
        will not be considered as relevant.</rdfs:comment>
        <rdfs:subClassOf
        rdf:resource="http://www.metarel.org/RelationalSubsumptionClosure"/>
</rdf:Description>
<rdf:Description rdf:about="http://www.metarel.org/RelevantRelationType">
        <rdfs:label>relevant relation type</rdfs:label>
        <rdfs:comment>A relevant relation type is a relation type that
        users of a knowledge base might need in their queries or in
        answers on queries. E.g. 'is related to' is not a relevant
        relation type.</rdfs:comment>
        <rdfs:subClassOf
        rdf:resource="http://www.metarel.org/RelationType"/>
</rdf:Description>
<rdf:Description
rdf:about="http://www.metarel.org/SemanticsProvidingConstruct">
        <rdfs:label>semantics-providing construct</rdfs:label>
        <rdfs:comment>A semantics-providing construct provides meaning
        to relation types in a Knowledge Base by indicating which
        relation arcs can be inferred in the Knowledge Base.
        </rdfs:comment>
</rdf:Description>
<rdf:Description
rdf:about="http://www.metarel.org/SomeSomeClassRelationType">
        <rdfs:label>some-some class relation type</rdfs:label>
        <rdfs:comment>A some-some class relation type is an
        instance-based class relation type of which all the relation
        arcs are allowed to be not total and are allowed to be not
        surjective. A class relation arc is total if all the instances
        of the source class have at least one corresponding instance
        relation arc to the target class. It is surjective if all the
        instances from the target class receive at least one
        corresponding instance relation arc from the source class. E.g.
        'human eats pig' is a some-some relation arc as some people do
        not eat pork, and not all pigs are cattle.</rdfs:comment>
        <rdfs:subClassOf rdf:resource=
        "http://www.metarel.org/InvertibleInstanceBasedClassRelationType"/>
        <rdfs:comment>The some-some relation type should be avoided in a
        good ontology design.</rdfs:comment>
</rdf:Description>
<rdf:Description rdf:about="http://www.metarel.org/SubjectDataRelationType">
        <rdfs:label>subject-data relation type</rdfs:label>
        <rdfs:comment>A subject-data relation type is a relation type of
        which all the relations are from a resource to a data type.
        </rdfs:comment>
        <rdfs:subClassOf
        rdf:resource="http://www.metarel.org/RelationType"/>
        <owl:sameAs
        rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</rdf:Description>
<rdf:Description
```

```
rdf:about="http://www.metarel.org/SubjectObjectRelationType">
        <rdfs:label>subject-object relation type</rdfs:label>
        <rdfs:comment>Any relation type of which all the relations are
        between two resources is a subject-object relation type.
        </rdfs:comment>
        <rdfs:subClassOf
        rdf:resource="http://www.metarel.org/RelationType"/>
        <owl:sameAs
        rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
</rdf:Description>
<rdf:Description rdf:about="http://www.metarel.org/SymmetricRelationType">
        <rdfs:label>symmetric relation type</rdfs:label>
        <rdfs:comment>A relation type R is a symmetric relation type
        when A R B implies B R A.</rdfs:comment>
        <rdfs:subClassOf
        rdf:resource="http://www.metarel.org/SubjectObjectRelationType"/>
        <owl:sameAs
        rdf:resource="http://www.w3.org/2002/07/owl#SymmetricProperty"/>
</rdf:Description>
<rdf:Description rdf:about="http://www.metarel.org/TightClassRelationType">
        <rdfs:label>tight class relation type</rdfs:label>
        <rdfs:comment>A tight class relation type is an instance-based
        class relation type of which all the class relation arcs should
        be total and surjective. A class relation arc is total and
        surjective if there starts at least one corresponding instance
        relation arc for every instance in the source class and if there
        arrives at least one corresponding instance relation arc for
        every instance in the target class. E.g. 'tusk is integral part
        of elephant' is a total and surjective class relation arc as
        every tusk is part of an elephant and for every elephant there
        is a tusk which is a part of the elephant.</rdfs:comment>
        <rdfs:subClassOf rdf:resource=
        "http://www.metarel.org/InvertibleInstanceBasedClassRelationType"/>
</rdf:Description>
<rdf:Description rdf:about="http://www.metarel.org/TransitiveClosure">
        <rdfs:label>transitive closure</rdfs:label>
        <rdfs:comment>A transitive closure is an inference mechanism
        that infers the resulting relation types of transitive
        composites. It implies the inference of all the relation arcs
        that follow from transitive relation types.</rdfs:comment>
        <rdfs:subClassOf
        rdf:resource="http://www.metarel.org/CompositionalClosure"/>
</rdf:Description>
<rdf:Description
rdf:about="http://www.metarel.org/TransitiveOverComposite">
        <rdfs:label>transitive-over composite</rdfs:label>
        <rdfs:comment>The resulting relation type is the same as the
        first relation type, but different from the second relation
        type. If the first and the second relation type change their
        positions, then it does not give the same resulting relation
        type any longer. E.g. 'is brother of' + 'is sister of' results
        in 'is brother of'. In DL notation%3A R o S %3C R.
        </rdfs:comment>
        <rdfs:subClassOf
        rdf:resource="http://www.metarel.org/RelationalComposite"/>
</rdf:Description>
<rdf:Description
rdf:about="http://www.metarel.org/TransitiveRelationType">
        <rdfs:label>transitive relation type</rdfs:label>
        <rdfs:comment>A relation type R is a transitive relation type
        when A R B and B R C implies A R C.</rdfs:comment>
        <rdfs:subClassOf
        rdf:resource="http://www.metarel.org/SubjectObjectRelationType"/>
        <owl:sameAs
        rdf:resource="http://www.w3.org/2002/07/owl#TransitiveProperty"/>
</rdf:Description>
<rdf:Description
rdf:about="http://www.metarel.org/TransitivityComposite">
        <rdfs:label>transitivity composite</rdfs:label>
        <rdfs:comment>The first, the second and the resulting relation
        type are all the same for a transitive composite, so this
        relation type is transitive. E.g. 'is ancestor of' + 'is
        ancestor of' results in 'is ancestor of'. In DL notation
        %3A R o R %3C R.</rdfs:comment>
```

```
        <rdfs:subClassOf
        rdf:resource="http://www.metarel.org/RelationalComposite"/>
</rdf:Description>
<rdf:Description rdf:about="http://www.metarel.org/TwinComposite">
        <rdfs:label>twin composite</rdfs:label>
        <rdfs:comment>The first and the second relation type are the
        same for a twin composite, but they differ from the resulting
        relation type E.g. 'is father of' + 'is father of' results in
        'is grandfather of'. In DL notation%3A R o R %3C S.
        </rdfs:comment>
        <rdfs:subClassOf
        rdf:resource="http://www.metarel.org/RelationalComposite"/>
</rdf:Description>
<metarel:metarelationType
rdf:about="http://www.metarel.org/hasTransitiveClosure">
        <rdfs:label>has transitive closure</rdfs:label>
        <rdfs:comment>'Has transitive closure' is a metarelation between
        two relation types. A relation type R is the transitive closure
        of a relation type S if and only if R is transitive, S is a R,
        there is no transitive relation type T, other than R, for which
        S is a T and R is a T.
        Then we have S has transitive closure R.</rdfs:comment>
        <rdfs:subClassOf
        rdf:resource="http://www.metarel.org/isMetaRelatedTo"/>
</metarel:metarelationType>
<metarel:metarelationType
rdf:about="http://www.metarel.org/hasResultingRelation">
        <rdfs:label>has resulting relation</rdfs:label>
        <rdfs:comment>'Has resulting relation' is a metarelation that
        relates a composite with a relation type. Consider the
        composite R_S_T, and the relation types R, S and T imply%3A if
        A is R-related to B and B is S-related to C, than A is T-related
        to C. Than R_S_T has resulting relation T.</rdfs:comment>
        <rdfs:subClassOf
        rdf:resource="http://www.metarel.org/isMetaRelatedTo"/>
</metarel:metarelationType>
<metarel:metarelationType
rdf:about="http://www.metarel.org/hasSecondRelation">
        <rdfs:label>has second relation</rdfs:label>
        <rdfs:comment>'Has second relation' is a metarelation that
        relates a composite with a relation type. Consider the composite
        R_S_T, and the relation types R, S and T imply%3A if A is
        R-related to B and B is S-related to C, than A is T-related to C.
        Than R_S_T has second relation S.</rdfs:comment>
        <rdfs:subClassOf
        rdf:resource="http://www.metarel.org/isMetaRelatedTo"/>
</metarel:metarelationType>
<metarel:metarelationType
rdf:about="http://www.metarel.org/isReciprocalOf">
        <rdfs:label>is reciprocal of</rdfs:label>
        <rdfs:comment>A class relation type A is the reciprocal of a
        class relation type B if A and B belong to the same type of
        class relation types (meaning they are based on instance
        relation types by the same quantification method) and if the
        instance relation type a is the inverse of the instance relation
        type b, where A is based on a and B is based on b.</rdfs:comment>
        <rdfs:subClassOf
        rdf:resource="http://www.metarel.org/isMetaRelatedTo"/>
</metarel:metarelationType>
<metarel:metarelationType rdf:about="http://www.metarel.org/isBasedOn">
        <rdfs:label>is based on</rdfs:label>
        <rdfs:comment>An instance-based relation type 'is based on' the
        corresponding instance relation type on which it is based. E.g.
        'is part of' (like in 'leg is part of body') is based on the
        instance-'is part of' (like in 'Belgium is part of Europe').
        </rdfs:comment>
        <rdfs:subClassOf
        rdf:resource="http://www.metarel.org/isMetaRelatedTo"/>
</metarel:metarelationType>
<metarel:metarelationType
rdf:about="http://www.metarel.org/buildsDefiningConditionFrom">
        <rdfs:label>builds defining condition from</rdfs:label>
        <rdfs:comment>For every instance-based relation type, it is
        possible to build defining relation types. The defining relation
```

```
                type 'builds defining condition from' the original
                instance-based relation type.</rdfs:comment>
                <rdfs:subClassOf
                rdf:resource="http://www.metarel.org/isMetaRelatedTo"/>
        </metarel:metarelationType>
        <metarel:metarelationType
        rdf:about="http://www.metarel.org/hasFirstRelation">
                <rdfs:label>has first relation</rdfs:label>
                <rdfs:comment>'Has first relation' is a metarelation that
                relates a composite with a relation type. Consider the composite
                R_S_T, and the relation types R, S and T imply%3A if A is
                R-related to B and B is S-related to C, than A is T-related to C.
                Than R_S_T has first relation R.</rdfs:comment>
                <rdfs:subClassOf
                rdf:resource="http://www.metarel.org/isMetaRelatedTo"/>
        </metarel:metarelationType>
        <metarel:metarelationType
        rdf:about="http://www.metarel.org/isMetaRelatedTo">
                <rdfs:label>is metarelated to</rdfs:label>
                <rdfs:comment>An 'is metarelated to' is a construct that
                relates relation types (the instances of the relation type
                classification), that have logical connections with each other.
                </rdfs:comment>
        </metarel:metarelationType>
        <metarel:metarelationType rdf:about="http://www.metarel.org/isInverseOf">
                <rdfs:label>is inverse of</rdfs:label>
                <rdfs:comment>Relation type A is the inverse of relation type B
                if for every relation that is an instance of A, the relation in
                the opposite direction is an instance of B.</rdfs:comment>
                <rdfs:subClassOf rdf:resource="http://www.metarel.org/isMetaRelatedTo"/>
        </metarel:metarelationType>
</rdf:RDF>
```

Appendix B

Biorel

Biorel is an ontology for relation types. It has assembled all the relation types that were used in any of the OBO Foundry ontologies, OBO Foundry candidate ontologies, the Relationship Ontology and CCO (October 2010). Many entries of relation types that were duplicates were compared and manually curated via spreadsheets, GNU sorting operations and a text editor. In this manner, 833 relation type entries were reduced in a consistent effort to 365 unique, curated relation types.

B.1 The OBO format

The curation effort has resulted directly in a valid OBO file that can be used in combination with any OBO ontologies that use the relation types in Biorel. This file, biorel.obo, was also used as an input file for the programmatic pipeline that creates CCO. The procedure for BioGateway is slightly more advanced, since biorel.obo was translated into OWL 2 DL before it was uploaded in the RDF store.

Here are included the first few entries of relation types in the OBO format, as well as some special cases:

biorel.obo:

```
format-version: 1.4
date: 05:10:2010 13:45
saved-by: Ward Blondé
auto-generated-by: ONTO-PERL 1.28

[Typedef]
id: activates
name: activates
is_a: positively_regulates ! positively regulates
```

```
[Typedef]
id: acts_on
name: acts on
def: "A relation between a process and a continuant, where the continuant plays
the role of the entity that is changed by that process. Instance: p acts_on c:
exists Quality q, Time t and q inheres_in c at t, such that the continual
unfolding of p results_in changes in q or maintenance of q; Type: all pP, exists
cCt such that p acts_on c" []
comment: DEPRECATED Examples: Organismal growth acts_on organism; neuron
migration acts_on neuron;
synonym: "unfolds_towards" RELATED []
synonym: "has_direct_participant" RELATED []

[Typedef]
id: acts_on_population_of
name: acts on population of
def: "This is a class-level relation only. If P acts_on_population_of C, then:
for all instances p of P, there exists some cp, t such that c instantiates CP at
t, p has_central_participant cp, and CP is equivalent to population_of(C). For
example: T-cell_proliferation acts_on_population_of T-cell - for all instances
of T-cell_proliferation have a population of T-cells as central_participants" []

[Typedef]
id: adheres_in
name: adheres in

.

.

.

[Typedef]
id: is_a
name: is
builtin: true
def: "For continuants: C is_a C' if and only if: given any c that instantiates C
at a time t, c instantiates C' at t. For processes: P is_a P' if and only if:
that given any p that instantiates P, then p instantiates P'." [PMID:15892874]
comment: The is_a relationship is considered axiomatic by the obo file format
specification, and by OWL
synonym: "is_subtype_of" RELATED []
xref: rdfs:subClassOf
is_anti_symmetric: true
is_reflexive: true
is_transitive: true

.

.

.

[Typedef]
id: integral_part_of
name: is integral part of
def: "C integral_part_of C' if and only if: C part_of C' AND C' has_part C"
[PMID:15892874]
is_anti_symmetric: true
is_reflexive: true
is_transitive: true
is_a: part_of ! is part of
inverse_of: has_integral_part ! has integral part
is_metadata_tag: true

.

.

.

[Typedef]
id: part_of
```

```
name: is part of
def: "For continuants: C part_of C' if and only if: given any c that instantiates
C at a time t, there is some c' such that c' instantiates C' at time t, and c
*part_of* c' at t. For processes: P part_of P' if and only if: given any p that
instantiates P at a time t, there is some p' such that p' instantiates P' at time
t, and p *part_of* p' at t. (Here *part_of* is the instance-level part-
relation.)" [PMID:15892874]
is_reflexive: true
is_transitive: true
inverse_of: has_part ! has part
holds_over_chain: results_in_formation_of ends_during
holds_over_chain: results_in_complete_development_of starts_during

.

.

.

[Typedef]
id: variant_of
name: is variant of
def: "A' is a variant (mutation) of A = definition every instance of A' is either
an immediate mutation of some instance of A, or there is a chain of immediate
mutation processes linking A' to some instance of A." [SO:immuno_workshop]
comment: Added to SO during the immunology workshop, June 2007.  This
relationship was approved by Barry Smith.
```

B.2 OWL 2 DL for validation and reasoning

The OBO format serves as the original for the translation into a Semantic Web compatible format. OWL is the best choice for this purpose. It was created to enable reasoning and it also has an RDF/XML syntax. There are several sublanguages of the complete set of OWL language constructs (OWL Full). The largest sublanguage that is still decidable (which is a warrant for usefulness towards automated reasoning) is OWL 2 DL, the successor of OWL DL.

The translation was validated with two validator tools on the web:

- **http://www.w3.org/RDF/Validator/**, a W3C validation service for RDF documents providing useful feedback on possible flaws in the RDF/XML syntax
- **http://owl.cs.manchester.ac.uk/validator/**, the Manchester validator that can unambiguously decide whether the representation is valid OWL 2 DL or not.

By choosing for OWL 2 DL, a maximum of logic-based semantics for the relation types was extracted from the OBO format and made available to a much wider community. The integration of this file, biorel.owl, with metarel.rdf in BioGateway will enable the five efficient relation closure rules for class relation types.

biorel.owl:

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:oboInOwl="http://purl.obolibrary.org/obo/oboInOwl#"
    xmlns:obo="http://purl.obolibrary.org/obo/"
>
<owl:AnnotationProperty
  rdf:about="http://purl.obolibrary.org/obo/oboInOwl#hasURI"/>
<owl:AnnotationProperty
  rdf:about="http://purl.obolibrary.org/obo/oboInOwl#hasAlternativeId"/>

.

.

.

<owl:Ontology rdf:about="">
    <oboInOwl:hasDate>05:10:2010 13:45</oboInOwl:hasDate>
    <oboInOwl:savedBy>Ward Blondé</oboInOwl:savedBy>
    <rdfs:comment>Biorel.obo assembles all the relationship types that are used
    in OBO ontologies. Many corrections have been carried out, in order to use
    Biorel for automated reasoning. The relationship types were given a name that
    contains a verb in the third person singular, to get a sound and clear
    meaning that can be understood by end-users of a query-system. All the
    relationship types in Biorel are assumed to operate between instances
    (instance relation types or owl:ObjectProperty's), except those that have the
    tag 'is_metadata_tag: true' (they are owl:AnnotationProperty's).
    </rdfs:comment>
</owl:Ontology>

<owl:ObjectProperty rdf:about="http://purl.obolibrary.org/obo/activates">
    <rdfs:label>activates</rdfs:label>
    <rdfs:subPropertyOf
    rdf:resource="http://purl.obolibrary.org/obo/positively_regulates"/>
</owl:ObjectProperty>

.

.

.

<owl:AnnotationProperty
rdf:about="http://purl.obolibrary.org/obo/integral_part_of">
    <rdfs:label>is integral part of</rdfs:label>
    <oboInOwl:hasDefinition>
        <oboInOwl:Definition>
            <rdfs:label>C integral_part_of C' if and only if%3A C part_of C' AND
            C' has_part C</rdfs:label>
            <oboInOwl:hasDbXref>
                <oboInOwl:DbXref>
                    <rdfs:label>PMID:15892874</rdfs:label>
                    <oboInOwl:hasURI
                    rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
                    http://PMID#PMID_15892874</oboInOwl:hasURI>
                </oboInOwl:DbXref>
            </oboInOwl:hasDbXref>
        </oboInOwl:Definition>
    </oboInOwl:hasDefinition>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#AnnotationProperty"/>
</owl:AnnotationProperty>

.

.

.
```

```
<owl:ObjectProperty rdf:about="http://purl.obolibrary.org/obo/part_of">
    <rdfs:label>is part of</rdfs:label>
    <oboInOwl:hasDefinition>
        <oboInOwl:Definition>
            <rdfs:label>For continuants%3A C part_of C' if and only if%3A given
            any c that instantiates C at a time t, there is some c' such that c'
            instantiates C' at time t, and c *part_of* c' at t. For processes%3A
            P part_of P' if and only if%3A given any p that instantiates P at a
            time t, there is some p' such that p' instantiates P' at time t, and
            p *part_of* p' at t. (Here *part_of* is the instance-level part-
            relation.)</rdfs:label>
            <oboInOwl:hasDbXref>
                <oboInOwl:DbXref>
                    <rdfs:label>PMID:15892874</rdfs:label>
                    <oboInOwl:hasURI
                    rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
                    http://PMID#PMID_15892874</oboInOwl:hasURI>
                </oboInOwl:DbXref>
            </oboInOwl:hasDbXref>
        </oboInOwl:Definition>
    </oboInOwl:hasDefinition>
    <owl:inverseOf rdf:resource="http://purl.obolibrary.org/obo/has_part"/>
    <owl:propertyChainAxiom rdf:parseType="Collection">
        <owl:ObjectProperty
        rdf:about="http://purl.obolibrary.org/obo/results_in_formation_of"/>
        <owl:ObjectProperty
        rdf:about="http://purl.obolibrary.org/obo/ends_during"/>
    </owl:propertyChainAxiom>
    <owl:propertyChainAxiom rdf:parseType="Collection">
        <owl:ObjectProperty rdf:about=
        "http://purl.obolibrary.org/obo/results_in_complete_development_of"/>
        <owl:ObjectProperty
        rdf:about="http://purl.obolibrary.org/obo/starts_during"/>
    </owl:propertyChainAxiom>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#TransitiveProperty"/>
</owl:ObjectProperty>

.

.

.

</rdf:RDF>

<!--
Generated with ONTO-PERL: obo2owl.pl, 21:06:2010 11:17
Debugged and extended manually by Ward Blondé, 04:10:2010 17:15
-->
```

Appendix C

BioMetarel

The creation of BioMetarel consists of three steps:

- Loading the RDF exports of Metarel and Biorel in the same RDF graph.
- Loading a manually maintained set of links between Biorel en Metarel, in the form of RDF triples.
- Inferring all the possible directly available conclusions about the semantics of the relation types through automated inference procedures.

C.1 The links between Metarel and Biorel

Metarel contains many terms that can be used to annotate the relation types in Biorel. These annotations form the links between Biorel and Metarel, which have to be created manually by somebody who can interpret the semantics of the relation types in Biorel. Metarel also provides the metarelations that can stand in between two relation types. Such manually curated annotations can be found in the RDF file biometarel_merge.rdf.turtle underneath. The usage of the turtle syntax for RDF, as opposed to the XML syntax, is very useful for operations in a text editor and it is correctly parsed and loaded by Virtuoso.

Not every link between Metarel en Biorel is included in this file. Most semantic features of the relation types are expressed in the OBO format and translated in the RDF export file biorel.rdf. They form the basis to

create links between Metarel and Biorel that are created automatically with SPARUL after the loading of the files metarel.rdf and biorel.rdf. The file biometarel_merge.rdf.turtle is used first of all to create a consistent distinction between class relation types and instance relation types. It is also used to formally relate the so-called 'built-in' *is_a* relation type in OBO with *rdfs:subClassOf* and *owl:sameAs* in the realm of the Semantic Web. Some missing annotations of reflexivity are also added here.

biometarel_merge.rdf.turtle:

```
@prefix ssb: <http://www.semantic-systems-biology.org/SSB#>.
@prefix metarel: <http://www.metarel.org/>.
@prefix obo: <http://purl.obolibrary.org/obo/>.
@prefix rdfs:<http://www.w3.org/2000/01/rdf-schema#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.

ssb:activates metarel:isBasedOn obo:activates.
ssb:acts_on metarel:isBasedOn obo:acts_on.
ssb:acts_on_population_of metarel:isBasedOn obo:acts_on_population_of.
ssb:adheres_in metarel:isBasedOn obo:adheres_in.
ssb:adjacent_to metarel:isBasedOn obo:adjacent_to.

.

.

.

ssb:unfolds_around metarel:isBasedOn obo:unfolds_around.
ssb:unfolds_in metarel:isBasedOn obo:unfolds_in.
ssb:unit_of metarel:isBasedOn obo:unit_of.
ssb:variant_of metarel:isBasedOn obo:variant_of.

ssb:activates rdf:type metarel:AllSomeClassRelationType.
ssb:acts_on rdf:type metarel:AllSomeClassRelationType.
ssb:acts_on_population_of rdf:type metarel:AllSomeClassRelationType.
ssb:adheres_in rdf:type metarel:AllSomeClassRelationType.
ssb:adjacent_to rdf:type metarel:AllSomeClassRelationType.

.

.

.

obo:activates rdf:type metarel:InstanceRelationType.
obo:acts_on rdf:type metarel:InstanceRelationType.
obo:acts_on_population_of rdf:type metarel:InstanceRelationType.
obo:adheres_in rdf:type metarel:InstanceRelationType.
obo:adjacent_to rdf:type metarel:InstanceRelationType.
```

```
.

.

.

ssb:is_a metarel:isBasedOn owl:sameAs.
ssb:is_a rdf:type metarel:AllSomeClassRelationType.
owl:sameAs rdf:type metarel:InstanceRelationType.
ssb:is_a owl:sameAs rdfs:subClassOf.
ssb:is_a rdfs:label "is".

ssb:integral_part_of metarel:isBasedOn obo:part_of.
ssb:integral_part_of rdf:type metarel:TightClassRelationType.
ssb:integral_part_of rdfs:label "is integral part of".
ssb:has_integral_part metarel:isBasedOn obo:has_part.
ssb:has_integral_part rdf:type metarel:TightClassRelationType.
ssb:has_integral_part rdfs:label "has integral part".

owl:sameAs rdf:type metarel:ReflexiveRelationType.
obo:part_of rdf:type metarel:ReflexiveRelationType.
obo:located_in rdf:type metarel:ReflexiveRelationType.
owl:sameAs rdf:type owl:TransitiveProperty.
```

C.2 The RDF export

After the loading of metarel.rdf, biorel.rdf and biometarel_merge.rdf.turtle, followed by the automated inferring of new triples with SPARUL, BioMetarel is created as a graph within the RDF store. Unfortunately it is not possible to create a nice RDF export that is somewhat human readable. With the following SPARQL command, Virtuoso can create an export in the RDF/XML syntax:

```
BASE   <http://www.semantic-systems-biology.org/>
CONSTRUCT {?s ?p ?o.}
WHERE {
 GRAPH <biometarel> {
  ?s ?p ?o.
 }
}
```

Every triple in the graph for BioMetarel is represented as a single line in the file, without any special formatting or ordering for readability.

This file can be used for the distribution of BioMetarel to other RDF stores. An abbreviated section is shown here:

biometarel.rdf:

```
<?xml version="1.0" encoding="utf-8" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:rdfs="http://www.w3.
<rdf:Description rdf:about="http://www.semantic-systems-biology.org/SSB#has_quality"><rdfs:
<rdf:Description rdf:nodeID="b1000010675"><rdf:rest rdf:resource="http://www.w3.org/1999/02
<rdf:Description rdf:about="http://purl.obolibrary.org/obo/executed_in"><rdf:type rdf:resou
<rdf:Description rdf:nodeID="b1000010780"><n0pred:hasURI xmlns:n0pred="http://purl.obolibra
<rdf:Description rdf:about="http://purl.obolibrary.org/obo/interacts_with"><rdf:type rdf:re
<rdf:Description rdf:nodeID="b1000010592"><rdfs:label>UBERON:cjm</rdfs:label></rdf:Descript
<rdf:Description rdf:about="http://purl.obolibrary.org/obo/proper_part_of"><rdfs:label>is p
<rdf:Description rdf:about="http://purl.obolibrary.org/obo/exemplar_of"><rdf:type rdf:resou
<rdf:Description rdf:nodeID="b1000010463"><n0pred:hasURI xmlns:n0pred="http://purl.obolibra
<rdf:Description rdf:about="http://purl.obolibrary.org/obo/inversely_associated_with"><rdf:
<rdf:Description rdf:about="http://www.semantic-systems-biology.org/SSB#results_in_fusion_o
<rdf:Description rdf:nodeID="b1000010778"><n0pred:hasURI xmlns:n0pred="http://purl.obolibra
<rdf:Description rdf:about="http://purl.obolibrary.org/obo/existence_starts_after"><owl:pro
<rdf:Description rdf:about="http://www.semantic-systems-biology.org/SSB#results_in_breakdow
<rdf:Description rdf:nodeID="b1000010537"><rdfs:label>catabolises into</rdfs:label></rdf:De
<rdf:Description rdf:about="http://purl.obolibrary.org/obo/existence_starts_with"><owl:prop
<rdf:Description rdf:nodeID="b1000010776"><n0pred:hasURI xmlns:n0pred="http://purl.obolibra
<rdf:Description rdf:about="http://purl.obolibrary.org/obo/has_relative_magnitude"><rdfs:la
<rdf:Description rdf:about="http://www.semantic-systems-biology.org/SSB#transforms_into"><r
<rdf:Description rdf:nodeID="b1000010588"><rdfs:label>UBERON:cjm</rdfs:label></rdf:Descript
<rdf:Description rdf:about="http://www.semantic-systems-biology.org/SSB#catalyses"><n0pred:
<rdf:Description rdf:about="http://purl.obolibrary.org/obo/is_measurement_of"><rdfs:label>i
<rdf:Description rdf:nodeID="b1000010587"><n0pred:hasDbXref xmlns:n0pred="http://purl.oboli
<rdf:Description rdf:about="http://www.semantic-systems-biology.org/SSB#qualifier"><n0pred:
<rdf:Description rdf:about="http://www.semantic-systems-biology.org/SSB#innervates"><rdfs:l
<rdf:Description rdf:nodeID="b1000010586"><rdfs:label>UBERON:cjm</rdfs:label></rdf:Descript
<rdf:Description rdf:about="http://www.semantic-systems-biology.org/SSB#has_regexp"><rdfs:l
<rdf:Description rdf:nodeID="b1000010585"><n0pred:hasDbXref xmlns:n0pred="http://purl.oboli
<rdf:Description rdf:about="http://www.metarel.org/DefiningRelationType"><rdfs:label>defini
<rdf:Description rdf:about="http://purl.obolibrary.org/obo/existence_starts_during"><owl:pr
<rdf:Description rdf:nodeID="b1000010773"><n0pred:hasURI xmlns:n0pred="http://purl.obolibra
<rdf:Description rdf:about="http://www.semantic-systems-biology.org/SSB#recombined_to"><n0p
<rdf:Description rdf:about="http://purl.obolibrary.org/obo/ends_earlier_than"><rdfs:label>e
<rdf:Description rdf:nodeID="b1000010580"><n0pred:hasDbXref xmlns:n0pred="http://purl.oboli
<rdf:Description rdf:about="http://purl.obolibrary.org/obo/paralogous_to"><rdfs:label>is pa
<rdf:Description rdf:nodeID="b1000010582"><n0pred:hasDbXref xmlns:n0pred="http://purl.oboli
<rdf:Description rdf:about="http://purl.obolibrary.org/obo/results_in_commitment_to"><rdfs:
<rdf:Description rdf:about="http://www.semantic-systems-biology.org/SSB#integral_part_of"><
<rdf:Description rdf:nodeID="b1000010863"><n0pred:hasURI xmlns:n0pred="http://purl.obolibra
<rdf:Description rdf:about="http://www.semantic-systems-biology.org/SSB#encoded_by"><rdfs:l
<rdf:Description rdf:about="http://www.semantic-systems-biology.org/SSB#translates_to"><rdf
<rdf:Description rdf:nodeID="b1000010591"><n0pred:hasDbXref xmlns:n0pred="http://purl.oboli
<rdf:Description rdf:about="http://purl.obolibrary.org/obo/negatively_regulates_timing_of">
<rdf:Description rdf:about="http://purl.obolibrary.org/obo/existence_starts_and_ends_during
<rdf:Description rdf:nodeID="b1000010551"><n0pred:hasDbXref xmlns:n0pred="http://purl.oboli
<rdf:Description rdf:about="http://www.semantic-systems-biology.org/SSB#results_in_remodeli
<rdf:Description rdf:nodeID="b1000010741"><rdf:rest rdf:nodeID="b1000010740"/></rdf:Descrip
<rdf:Description rdf:nodeID="b1000010803"><rdfs:label>results_in_targeting_to_from_or_acros
<rdf:Description rdf:about="http://www.semantic-systems-biology.org/SSB#results_in"><rdf:ty
<rdf:Description rdf:nodeID="b1000010494"><rdfs:label>Near the outer surface of the organis
<rdf:Description rdf:about="http://www.semantic-systems-biology.org/SSB#has_central_partici
<rdf:Description rdf:about="http://www.semantic-systems-biology.org/SSB#results_in_transpor
<rdf:Description rdf:about="http://www.semantic-systems-biology.org/SSB#has_axis"><n0pred:i
<rdf:Description rdf:nodeID="b1000010545"><n0pred:hasDbXref xmlns:n0pred="http://purl.oboli
<rdf:Description rdf:nodeID="b1000010441"><rdfs:label>x aco y iff x dco y OR x aco z and z
<rdf:Description rdf:about="http://www.semantic-systems-biology.org/SSB#results_in_addition

.

.

.

</rdf:RDF>
```

Appendix D

Reasoner code

This appendix contains the software code (a PERL program and SQL code) that executes the reasoning on BioGateway with BioMetarel. It also does the loading of the RDF files into the special architecture of BioGateway, but it assumes that these RDF files were previously created. This means that it executes a remodularized part of the Rdfer pipeline that was once written by Erick Antezana to load BioGateway [183]. The PERL program consists of only about 1000 lines, but its function is to write SQL code that can be interpreted by Virtuoso. This SQL code is a large file of around 10 MB in size and contains the detailed instructions for loading and reasoning.

D.1 The PERL program

The configuration variables for the PERL script below are set to the idealized practice. In reality, Virtuoso could not handle the full SQL script of 10 MB. By running the PERL script twice and altering the configuration variables, the recreation of BioGateway could be split into two parts. This whole process costs about two days and a half.

Vladimir Mironov contributed a lot to this code by creating many useful modules, which has made my original script much shorter and more reusable. He also gave it a nicer formatting. Some essential Virtuoso commands were copied from Erick Antezana's script.

```
#!/usr/bin/perl -w

use warnings;
use strict;
use Carp;
use DBI;
```

```
#####################################################
#
###  Global variables
###  these variables don't change in the course of execution
#
#####################################################

my $dba_user       = "***";
my $dba_pass       = "***";
my $connect_string = "dbi:ODBC:VirtuosoSSB";
my $prefix         = "http://www.semantic-systems-biology.org/";
my $ns             = $prefix . "SSB";

### Configuration variables - important to set properly!!!
my $use_dbi                        = 0; # set to 1 to use DBI for uploading
my $make_cleanup                   = 1; # set to 1 to clean the RDF store
my $create_inhouse_graphs          = 1; # 0 - no biometarel, bfo and spin
my $make_upload                    = 1; # 0 - skip uploading files into graphs
my $make_upload_tc                 = 0; # 1 - closures with ONTO-PERL
my $make_transitive_closures       = 1; # the number of iterations:
                                        # 0 - no SPARUL closures
                                        # 1 - SPARUL's recursion (Virt 5.0.8)
                                        # 45 - a safe depth for BioGateway
my $make_reflexive_closures        = 1; # 0 - no closures
my $make_superrelation_closures    = 1; # 0 - no closures
my $make_priority_over_isa_closures = 1; # 0 - no closures
my $make_compositional_closures    = 1; # 0 - no closures
my $make_obo_closures              = 1; # 0 - no closures
my $make_ncbi_closures             = 1; # 0 - no closures
my $make_goa_closures              = 1; # 0 - no closures
my $create_metaonto                = 1; # 0 - metaonto is not created

.

.

.

#####################################################
#
#    Main
#
#####################################################

my $dbh = DBI->connect( $connect_string, $dba_user, $dba_pass, { RaiseError => 1 } )
          if $use_dbi;

my @obo_graph_names    = get_graph_names($obo_data_path);
my @goa_graph_names    = get_graph_names($goa_data_path);

.

.

.

if ($make_cleanup) {
  chomp(my $date = `date` );
  print PIPELINE_LOG "\n\n\n START OF CLEANING THE STORE ($date):\n\n\n";
  print PIPELINE_SQL "delete from DB.DBA.RDF_QUAD;\n\n";
}


if ($create_inhouse_graphs) {
  # Creating the Biorel, Metarel, BioMetarel, bfo and spin graphs
  create_inhouse_graphs ();
}

if ($make_upload) {

  # Uploading the OBO_ontologies graphs:
  upload_dir( $obo_data_path, '1', "", 'SSB', 'OBO' );
  copy_triples(@obo_graph_names, 'biometarel');
```

```
  # Uploading the NCBI graphs
  add_triples( $ncbi_file, 'SSB', 'SSB_tc', 'ncbi' );
  copy_triples('ncbi', 'biometarel');

  # Uploading the UniProt graphs
  # add_triples( $trembl_file, 'SSB', 'SSB_tc', 'uniprot_trembl' ); # vlmir
  # add_triples( $biorel_file, 'uniprot_trembl' );                  # vlmir
  add_triples( $sprot_file,  'SSB', 'SSB_tc', 'uniprot_sprot' );
  copy_triples('uniprot_sprot', 'biometarel');

  # Uploading owl ontologies (merged-obi)
  upload_dir( $owl_data_path, '1', "", 'SSB', 'OWL' );

  # Uploading opengalen
  upload_dir( $opengalen_data_path, '1', "", 'SSB', 'OWL', 'opengalen' );
  upload_dir( $opengalen_data_path, '0', "", 'opengalen' );

  # Uploading the GOA graphs
  upload_dir( $goa_data_path, '1', "", 'SSB' ) ;
  copy_triples(@goa_graph_names, 'biometarel');

  copy_triples('SSB', 'SSB_tc', 'OBO', 'OBO_tc', 'biometarel');
}

### Adding closures

# Creating closures for OBO ontologies
if ($make_obo_closures) {
  upload_dir ($obo_tc_data_path, '1', "_tc")
    if $make_upload_tc;
  upload_dir( $obo_data_path, '1', "_tc" )
    if !$make_upload_tc; # no closures yet - vlmir
  add_reflexive_closures (@obo_tc_graph_names) if $make_reflexive_closures;
  my $count = 3;
  while ($count) {
  add_transitive_closures ($make_transitive_closures, @obo_tc_graph_names)
    if $make_transitive_closures; # the first arg is the number of iterations
  add_superrelation_closures (@obo_tc_graph_names)
    if $make_superrelation_closures;
  add_priority_over_isa_closures (@obo_tc_graph_names, "")
    if $make_priority_over_isa_closures;
  add_compositional_closures (@obo_tc_graph_names, "")
    if $make_compositional_closures;
  $count --;
  }
  map {copy_triples ('SSB_tc', 'OBO_tc', $_)} @obo_tc_graph_names;
  copy_triples( @obo_tc_graph_names, 'biometarel' );
}

# create closures for NCBI taxonomy
if ($make_ncbi_closures) {
  upload_dir( $ncbi_data_path, '1', "_tc" );  # no closures yet
  add_transitive_closures ($make_transitive_closures, 'ncbi_tc')
    if $make_transitive_closures; # the first arg is the number of iterations
  add_reflexive_closures ('ncbi_tc')
    if $make_reflexive_closures;
  add_superrelation_closures ('ncbi_tc')
    if $make_superrelation_closures;
  add_priority_over_isa_closures ('ncbi_tc', "")
    if $make_priority_over_isa_closures;
  add_compositional_closures ('ncbi_tc', "")
    if $make_compositional_closures;
  map {copy_triples ('SSB_tc', $_)} 'ncbi_tc';
  copy_triples( 'ncbi_tc', 'biometarel' );
}


# Create closures for GOA
# (this does the work also for UniProt - for all proteins that are in GOA)
if ($make_goa_closures) {
  upload_dir( $goa_data_path, '1', "_tc" ); # no closures yet
  add_compositional_closures (@goa_tc_graph_names, 'gene_ontology_edit_tc')
    if $make_compositional_closures;
  add_priority_over_isa_closures ( @goa_tc_graph_names, 'gene_ontology_edit_tc')
```

```
    if $make_priority_over_isa_closures;
  add_priority_over_isa_closures ( @goa_tc_graph_names, 'ncbi_tc')
    if $make_priority_over_isa_closures;
  map {copy_triples ('SSB_tc', $_)} @goa_tc_graph_names;
  copy_triples( @goa_tc_graph_names, 'biometarel' );
}

# Create metaOnto, which contains some meta-information about the loaded store
if ($create_metaonto) {
  add_triples ($metaonto_file, 'metaonto');
  copy_triples ('metaonto', 'biometarel');
  insert_triples8 ('metaonto');
  insert_triples9 ('metaonto');
}
```

D.2 The SQL code

The SQL code essentially consists of a series of loading commands and SPARUL commands that will serve as input for a sequential execution by the Virtuoso software instance that will later answer the SPARQL queries. SPARUL is very similar to SPARQL, but it requires different execution privileges for security reasons.

Included below are the first lines of the SQL file that is used for creating BioGateway. The first commands serve for the creation of BioMetarel. After that starts the uploading of OBO ontologies into BioGateway.

```
delete from DB.DBA.RDF_QUAD;

sparql clear graph 'http://www.semantic-systems-biology.org/biorel';
sparql clear graph 'http://www.semantic-systems-biology.org/metarel';
sparql clear graph 'http://www.semantic-systems-biology.org/biometarel';
sparql clear graph 'http://www.semantic-systems-biology.org/bfo';

DB.DBA.RDF_LOAD_RDFXML_MT(file_to_string_output('/norstore/project/ssb/users/
ward/workspace/svn/data/rdf/inhouse/biorel.owl'),'http://www.semantic-systems-
biology.org/SSB','http://www.semantic-systems-biology.org/biorel');

DB.DBA.RDF_LOAD_RDFXML_MT(file_to_string_output('/norstore/project/ssb/users/
ward/workspace/svn/data/rdf/inhouse/metarel.rdf'),'http://www.semantic-systems-
biology.org/SSB','http://www.semantic-systems-biology.org/metarel');

.

.

sparql
BASE   <http://www.semantic-systems-biology.org/>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl:<http://www.w3.org/2002/07/owl#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX ssb:<http://www.semantic-systems-biology.org/SSB#>
PREFIX obo:<http://purl.obolibrary.org/obo/>
PREFIX metarel:<http://www.metarel.org/>
INSERT INTO GRAPH <biometarel> {
  ?node1 rdfs:subClassOf ?node3.
}
WHERE {
  GRAPH <biometarel> {
    ?node1 rdfs:subClassOf ?node2.
    ?node2 rdfs:subClassOf ?node3.
  }
}
;
```

```
.

.

sparql
# Propagate chains between instance relation types to the level of class relation
# types.
BASE   <http://www.semantic-systems-biology.org/>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl:<http://www.w3.org/2002/07/owl#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX ssb:<http://www.semantic-systems-biology.org/SSB#>
PREFIX obo:<http://purl.obolibrary.org/obo/>
PREFIX metarel:<http://www.metarel.org/>
INSERT INTO GRAPH <biometarel> {
  _:construct metarel:hasFirstRelation ?First_RelationType.
  _:construct metarel:hasSecondRelation ?Second_RelationType.
  _:construct metarel:hasResultingRelation ?Resulting_RelationType.
}
WHERE {
 GRAPH <biometarel> {
  ?Resulting_InstanceRelationType owl:propertyChainAxiom ?node.
  ?node rdf:first ?First_InstanceRelationType.
  ?node rdf:rest ?rest.
  ?rest rdf:first ?Second_InstanceRelationType.
  ?First_RelationType metarel:isBasedOn ?First_InstanceRelationType.
  ?Second_RelationType metarel:isBasedOn ?Second_InstanceRelationType.
  ?Resulting_RelationType metarel:isBasedOn ?Resulting_InstanceRelationType.
  ?First_RelationType rdf:type metarel:AllSomeClassRelationType.
  ?Second_RelationType rdf:type metarel:AllSomeClassRelationType.
  ?Resulting_RelationType rdf:type metarel:AllSomeClassRelationType.
 }
}
;

.

.

DB.DBA.TTLP_MT(file_to_string_output('/norstore/project/ssb/users/ward/
workspace/svn/data/rdf/inhouse/spin_closures/superrelation_closure.rdf.turtle'),
'http://www.semantic-systems-biology.org/SSB',
'http://www.semantic-systems-biology.org/spin');

DB.DBA.TTLP_MT(file_to_string_output('/norstore/project/ssb/users/ward/
workspace/svn/data/rdf/inhouse/spin_closures/relevant_relation_closure.rdf.turtle'),
'http://www.semantic-systems-biology.org/SSB',
'http://www.semantic-systems-biology.org/spin');

.

.
```

Appendix E

Scientific CV

Personal

Name: Ward
Middle names: Arthur Mark Carlo
Last name: Blondé
Citizenship: Belgian
Date of birth: 14 October 1979
E-mail: ward.blonde@ugent.be
Phone: +32/ 494 99.60.81
Dept. of Mathematical Modelling, Statistics and Bioinformatics
Ghent University
Coupure links 653
9000 Gent, Belgium

Research career

September 2011 until present:
Inst. of Medical Informatics, Statistics and Documentation
Medical University of Graz
LKH-Eingangszentrum, 3. and 5. Floor, Auenbruggerplatz 2, 8036 Graz, Austria

June 2009 - August 2011: continuation of PhD in Ghent, Belgium

February 2009 – May 2009: ESF project at NTNU in Trondheim, Norway.

January 2009: continuation of PhD at KERMIT:
Dept. of Mathematical Modelling, Statistics and Bioinformatics
Coupure links 653, Ghent University, 9000 Ghent

September 2006: start of PhD
Ph.D. student in Applied Biological Sciences
Project: An ontology for biomedical relations in the Semantic Web.
Promotors: Prof. dr. Martin Kuiper and Prof. dr. Bernard De Baets
Computational Biology Division
Plant Systems Biology
VIB/ Ghent University
9052 Zwijnaarde, Belgium

Degrees

2003-2005
Master in Biomedical and clinical engineering techniques
Two years program
Promotor: Prof. dr. ir. Carlos De Wagter
Ghent University

1997-2003
Civil Engineer Physics
Formal degree: Burgerlijk natuurkundig ingenieur
Minor option Biomedical techniques
Five years program
Promotor: Prof. dr. ir. Gert de Cooman
Ghent University

Language skills

Dutch: native tongue
English: excellent
French: good

German: good
Spanish: notions
Norwegian: notions

Experience

My research interests are biomedical relations and data integration with Semantic Web technologies. I have built a hierarchically organized vocabulary for biomedical relation types, Metarel, that builds a bridge between the OBO format and OWL, the current Semantic Web standard. I engineered and optimized BioGateway, an RDF repository that integrates biomedical ontologies like the Gene Ontology with genomic databases, like GOA and Uniprot. My acquired knowledge of OBO, RDF and SPARQL allowed me to approach wet-lab biologists to help them solving their problems from the computational side.
The construction of Metarel directed me towards applying semi-automated reasoning on RDF stores. This in turn brought me into contact with the fully automated reasoning approaches of OWL and Description Logics. I transformed the relation types in all the OBO ontologies to a single valid OWL2 DL relation ontology, called Biorel.

Journal Publications

V. Mironov, N. Seethappan, W. Blondé, E. Antezana, A. Splendiani, and M. Kuiper, "Gauging triple stores with actual biological data," *BMC Bioinformatics*, vol. 13, no. Suppl 1, pp. S3+, 2012.

W. Blondé, V. Mironov, A. Venkatesan, E. Antezana, B. De Baets, and M. Kuiper, "Reasoning with bio-ontologies: using relational closure rules to enable practical querying," *Bioinformatics*, vol. 27, pp. 1562–1568, 2011.

V. Mironov, E. Antezana, M. Egaña, W. Blondé, B. De Baets, R. Stevens, and M. Kuiper, "Flexibility and Utility of the Cell Cycle Ontology," *Applied Ontology*, vol. 6, no. 3, pp. 247–261, 2011.

E. Antezana, W. Blondé, M. Egaña, A. Rutherford, R. Stevens, B. De Baets, V. Mironov, and M. Kuiper, "BioGateway: a semantic systems biology tool

for the life sciences.," *BMC Bioinformatics*, vol. 10 Suppl 10, no. Suppl 10, pp. S11+, 2009.

E. Antezana, M. Egaña, W. Blondé, A. Illarramendi, I. Bilbao, B. De Baets, R. Stevens, V. Mironov, and M. Kuiper, "The Cell Cycle Ontology: An application ontology for the representation and integrated analysis of the cell cycle process," *Genome Biology*, vol. 10, no. 5, pp. R58+, 2009.

Submitted

A. Andrade, W. Blondé, J. Hastings, and S. Schulz, "Process attributes in bio-ontologies: accurate description of heart cycles.," *BMC Bioinformatics*

W. Blondé, E. Antezana, V. Mironov, S. Schulz, M. Kuiper and B. De Baets, "Using the relation ontology Metarel for modelling Linked Data as multi-digraphs." *Semantic Web Journal*, Special Issue

A. Venkatesan, W. Blondé, E. Antezana, S. Marshall, A. Splendiani, M. Egaña, J. Malone, V. Mironov and M. Kuiper, "Facilitating integrated analysis of biological data by enhancing interoperability of RDF resources: Practical Recommendations." *Semantic Web Journal*, Special Issue

Conference papers

A. Venkatesan, W. Blondé, E. Antezana, M. Skillingstad, M. S. Marshall, B. De Baets, V. Mironov, and M. Kuiper, "The RDF foundry: call for an initiative to build enhanced RDF resources for biological data integration," in *Proceedings of the International Conference on Web Intelligence, Mining and Semantics (WIMS '11)*, (New York, NY, USA), pp. 59:1–59:5, ACM, 2011.

E. Antezana, W. Blondé, A. Venkatesan, B. De Baets, V. Mironov, and M. Kuiper, "Semantic systems biology: enabling integrative biology via semantic web technologies," in *Proceedings of the International Conference on Web Intelligence, Mining and Semantics (WIMS '11)*, (New York, NY, USA), pp. 58:1–58:5, ACM, 2011.

V. Mironov, N. Seethappan, W. Blondé, E. Antezana, B. Lindi, and M. Kuiper, "Benchmarking triple stores with biological data," *CoRR*, vol. abs/1012.1632, 2010.

W. Blondé, E. Antezana, B. De Baets, V. Mironov, and M. Kuiper, "Metarel: an Ontology to support the inferencing of Semantic Web relations within Biomedical Ontologies," in *Proc. of the International Conference on Biomedical Ontologies (ICBO)*, pp. 79–82, 2009.

E. Antezana, W. Blondé, M. Egaña, A. Rutherford, R. Stevens, B. De Baets, V. Mironov, and M. Kuiper, "Structuring the life science resourceome for semantic systems biology: lessons from the BioGateway Project," in *Semantic Web Application and Tools 4 Life Science (SWAT4LS), Burger A, Paschke A, Romano, et al, eds*, vol. 435, 2008.

Presentations and posters at conferences

Metarel, a vocabulary for reasoning with biomedical relations.
From Nucleotides to Networks (N2N), Ghent, May 2011.

Metarel, a relation metagraph for inferences in biomedical ontologies.
European Conference on Computational Biology (ECCB 10), Ghent, Belgium, Sep. 2010.

Metarel: a Meta-Ontology for Reasoning with Biomedical Relations.
Reasoning Web Summer School, Brixen-Bressanone, Italy, Aug. 2009.

Metarel: an Ontology to support the inferencing of Semantic Web relations within Biomedical Ontologies.
International Conference on Biomedical Ontologies (ICBO), Buffalo, US, July 2009.

The Cell Cycle Ontology: an application ontology for data integration.
International Conference on Biomedical Ontologies (ICBO), Buffalo, US, July 2009.

BioGateway: enabling Semantic Systems Biology.

Bioinformatics Forum for Young Scientists (BFYS), Trondheim/Selbu, Norway, Apr. 2009.

BioGateway: integrated RDF for life science queries.
Semantic Web Applications and Tools for Life Sciences (SWAT4LS), Edinburgh, Scotland, UK., Nov. 2008.

BioGateway: enabling Semantic Systems Biology.
The 2nd International Workshop on Machine Learning in Systems Biology (MLSB), Brussels, Belgium, Sep. 2008.

CCO: an application ontology for the cell cycle.
Summer School on Ontological Engineering and Semantic Web (SSSW07), Madrid, Spain, July 2007.