UPTEC X 10 014

Examensarbete 30 hp
Juni 2010

SWI-Prolog as a Semantic Web Tool for semantic querying in Bioclipse: Integration and performance benchmarking

Samuel Lampa

UPPSALA
UNIVERSITET

Molecular Biotechnology Programme

Uppsala University School of Engineering

UPTEC X 10 014	**Date of issue 2010-07**
Author **Samuel Lampa**	
Title (English) **SWI-Prolog as a Semantic Web Tool for semantic querying in Bioclipse: Integration and performance benchmarking**	
Abstract The huge amounts of data produced in high-throughput techniques in the life sciences and the need for integration of heterogeneous data from disparate sources in new fields such as Systems Biology and translational drug development, require better approaches to data integration. The semantic web is anticipated to provide solutions through new formats for knowledge representation and management. Software libraries for semantic web formats are becoming mature, but there exist multiple tools based on foundationally different technologies. SWI-Prolog, a tool with semantic web support, was integrated into the Bioclipse bio- and cheminformatics workbench software and evaluated in terms of performance against non-Prolog-based semantic web tools in Bioclipse, Jena and Pellet, for querying a data set consisting of mostly numerical, NMR shift values, in the semantic web format RDF. The integration has given access to the convenience of the Prolog language for working with semantic data and defining data management workflows in Bioclipse. The performance comparison shows that SWI-Prolog is superior in terms of performance over Jena and Pellet for this specific dataset and suggests Prolog-based tools as interesting for further evaluations.	
Keywords Semantic Web, Prolog, Bioclipse, RDF, SPARQL, NMR shift, Eclipse, Java	
Supervisors **Dr. Egon Willighagen** **Uppsala University**	
Scientific reviewer **Prof. Mats Gustafsson** **Uppsala University**	
Project name	Sponsors
Language **English**	Security
ISSN 1401-2138	Classification
Supplementary bibliographical information	Pages **37**
Biology Education Centre Biomedical Center Husargatan 3 Uppsala Box 592 S-75124 Uppsala Tel +46 (0)18 4710000 Fax +46 (0)18 471 4687	

SWI-Prolog as a Semantic Web Tool for semantic querying in Bioclipse: Integration and performance benchmarking

Samuel Lampa

Sammanfattning

De så kallade livsvetenskaperna står inför enorma utmaningar beträffande datahantering. Orsaken är att det kommit flera nya tekniker inom området som producerar enorma mängder data, samt flera nya ämnesöverbryggande forskningsfält där man försöker integrera information från forskningsfält med vitt skilda kulturer och sätt att hantera data. Semantiska webben är ett koncept som anses kunna hjälpa till att effektivt möta en del av dessa utmaningar genom smartare sätt att representera och lagra data och kunskap. Det finns redan flera fullmogna programvaror för semantic web dataformat men det råder en uppdelning bland verktygen mellan fundamentalt olika grundteknologier. I den här studien har vi studerat ett semantic webbverktyg baserat på logikprogramspråket Prolog mot två verktyg baserat på programspråket Java. De två Javabaserade verktygen fanns integrerade i bioinformatikdataprogrammet Bioclipse och därför integrerades även det Prologbaserade verktyget i Bioclipse. När detta var gjort testades prestandan för de tre verktygen genom att utföra en semantisk fråga emot ett dataset mestadels bestående av numeriska data. Prestandan visade sig vara störst hos det Prologbaserade verktyget för detta dataset. Integrationen av det Prologbaserade semantic web verktyget har också gett tillgång till kraftfullheten i Prolog för att arbeta med semantiska data och definiera arbetsflöden för datahantering i Bioclipse.

Examensarbete 20p

Civilingenjörsprogrammet Molekylär bioteknik

Uppsala universitet Juni 2010

Contents

1 Introduction **2**

2 Background **3**
- 2.1 Data integration – a challenge for the life sciences . . . 3
- 2.2 The semantic web . . . 4
  - 2.2.1 RDF . . . 5
  - 2.2.2 RDF Schema (RDFS) . . . 6
  - 2.2.3 Web Ontology Language (OWL) . . . 6
  - 2.2.4 SPARQL . . . 7
  - 2.2.5 Reasoning . . . 7
  - 2.2.6 Rule languages . . . 8
- 2.3 Tools for managing semantic data . . . 8
  - 2.3.1 Importance of performance of tools in Bioclipse . . . 9
  - 2.3.2 Non-Prolog-based tools . . . 9
  - 2.3.3 Prolog . . . 9
- 2.4 Research question and project aims . . . 12

3 Methods **13**
- 3.1 Used Open Standards . . . 13
- 3.2 Used Open Source software . . . 13
- 3.3 Used Open Data . . . 15
- 3.4 Performance comparison . . . 18
- 3.5 MyExperiment website . . . 19

4 Results **21**
- 4.1 Implementation of research methods . . . 21
  - 4.1.1 Availability and requirements . . . 24
- 4.2 Performance comparison: SWI-Prolog vs Jena vs Pellet . . . 25

5 Discussion **30**
- 5.1 Integration of SWI-Prolog in Bioclipse . . . 30
- 5.2 Performance comparison . . . 30
- 5.3 Convenience of the Prolog language . . . 31
- 5.4 Implications for data integration . . . 32
- 5.5 Limitations of SWI-Prolog compared with Jena and Pellet . . . 32
- 5.6 Using Bioclipse for increased reproducibility . . . 32

6 Conclusion **33**

7 Acknowledgments **33**

1 Introduction

This project is about integrating a Prolog-based semantic web tool into the life science workbench Bioclipse, and comparing it against other, non-Prolog-based, semantic web tools available in Bioclipse, in order to highlight differences between these two kinds of technologies, foremost in terms of performance. For the comparison, a use case is addressed which is typical to the intended use of Bioclipse, involving data characterized by a relatively simple structure and with large quantities of similar items, of which most are numerical values.

The context of the study is thus the semantic web and related technologies, while the scope is the evaluation of currently available tools for managing semantic data, more specifically the comparison of a Prolog-based approach with a non-Prolog-based one, as being integrated in the Bioclipse workbench, and focusing on performance. The focus on performance is relevant in Bioclipse, since Bioclipse is focused on interactive use rather than the creation of automated workflows, which is the most typical focus for similar tools on the market. This will be more explained in the background section.

The appreciation of the relevance of this study requires some background both about the context of the study and of similar work within the scope of the research question. The report will therefore start with a discussion of the background that motivates the use of semantic web technologies in the first place, namely the challenges and need for better data integration in highly data intensive fields such as systems biology and translational drug development. It will then continue with an overview of earlier comparisons of semantic web tools, with the tools as independent applications, since no evaluation has previously been done with the tools integrated into Bioclipse.

Finally, the research question and project aims specific to this study will be presented, followed by the results section, which describes the integration of SWI-Prolog in Bioclipse and the results from the performance evaluations against other semantic web tools. The results section is followed by the discussion and conclusion sections.

2 Background

This section will cover the theory behind the semantic web and related technologies and the data integration challenges in the life sciences which motivate the use of semantic web technologies in the life sciences in the first place. The scope of the study will then be presented and finally the research question and project aims.

2.1 Data integration – a challenge for the life sciences

Data integration is a topic of increasing focus in the life sciences and bioinformatics communities. It is crucial to the success of new fields such as systems biology and translational drug development which rely on data and knowledge from diverse fields within molecular biology and biochemistry for its operation [1].

Data integration can mean many things in terms of practical scenarios, so let's consider a concrete scenario as an example. One of the "grand visions" for the semantic web is to be able to perform queries that automatically use multiple information sources to give the answer [2]. An example scenario of this kind, is the following (adapted from [2]): One might start with a set of yeast two hybrid protein-protein interactions, or list of highly interacting proteins, and then use this list to query the Entrez gene database [3] for the related gene names, resulting in a list of gene names. After the list of gene names is received, the Ensembl database [4] might be used for converting these to exon coordinates, which can in turn be checked for conservation scores using a tool such as UCSC Genome browser [5], with the list of genes annotated with conservation scores as result [2]. The automation of this kind of queries might serve as a foundation for thinking practically about the data integration challenges discussed next.

Many fields of the life sciences are today meeting exceptional challenges related to data integration. Challenges arise from the increasingly huge amounts of data produced [6] but even more from the heterogeneity of the data [7, 8]. The vast data amounts stem in part from the increasing use of high-throughput techniques such as DNA micro-array experiments and next generation sequencing [9]. The current pace of data production is so fast that scientists can no longer analyze the data in the same pace as new data is pouring in [10], which actualizes the need for better approaches to automated knowledge management and integration. For example, in order to keep up with the pace and annotate all new data pouring in, it might be needed to perform thousands of the kind of queries in the example scenario above, each day, or hour. This easily creates an untenable situations if the queries can not be executed automatically.

The heterogeneity of data in the life sciences can be exemplified with the highly varying nature of the entities described, including sequences, genes, proteins, pathways and drugs, all with intrinsically different means of representation. Adding to the challenges is that a variety of formats and standards often exist even for the same type of data, not to mention custom lay-outed spreadsheets or word processor documents completely lacking a formalized representation. In the example scenario, one can understand that the software that should automate it, needs to precisely understand all the different data formats used to describe the different entities involved, and be able to convert between them. And as soon as an additional data source is to be added, the software needs be adapted to take care of the intricacies specific to the new data source.

The data integration issues in the life sciences have already been addressed by a number of approaches. Examples include standardizations of experiment procedures, data exchange formats [11] and terminology, as well as wiki systems for community collaboration on integrated data sources [12]. *Standardization* of the data producing processes and the expression of the knowledge and data is vital [8] since it enables treating data from different sources in a uniform way. Standardization efforts in the life sciences have focused on applications such as protocols for describing biological experiments, XML-based data exchange formats and not the least the increasing adoption of bio-ontologies [6], which support knowledge integration by standardization of the terminology within a specific domain and its corresponding *meaning*. *Wiki systems* allow collaboration on data curation by many people on the same data, typically through the world wide web. This can make it realistic to create large integrated data sources manually.

While these efforts have facilitated data integration by increasing interoperability of data and allowing large scale collaboration efforts, they are still far from enabling full automation of data integration, for example so that the automation of the example scenario in the beginning of this section could be realized. Integration has still required detailed knowledge of the schema (a formalized description of the data structure) of each data source and the construction of mappings between the schemas for the source databases and the schema used for presentation, for example [13]. This situation is expected to come to a change with the advent of the *semantic web*, in part since the semantic web technologies remove the need for a fixed schema for each data source, as will be discussed in the next section. This dramatically lessens the need for cooperation between data providers for integration to happen [2] and thus removes a large barrier to automation of data integration.

The kind of information integration happening in the example scenario above, also stresses the demands for high performance in each of the steps, because, since it is typically organized more or less in a *series*, any delays in any of these steps will ultimately increase the total time for answering a query, due to the lack of parallelization. If the query is part of an interactive workflow, where the user is waiting for the results of a query before he/she can proceed with the next step, the demands become even more urgent, as will be discussed in more detail in subsubsection 2.3.1. This is the background for the focus on performance in this study, since the tools studied are indeed supposed to play a part in such interactive workflows.

2.2 The semantic web

The semantic web is an ongoing effort to alleviate integration of disparate data sources on the web through development of standards, formats and ontologies that enable better machine readability and hence more automation and computational assistance in data integration [14]. The concept of a "semantic web" was originally envisioned by Sir Tim

```
<http://saml.rilspace.com/> <http://purl.org/dc/elements/1.1/title> "Samuel's Project Blog" .
<http://saml.rilspace.com/> <http://www.example.org/owner> "Samuel Lampa" .
```

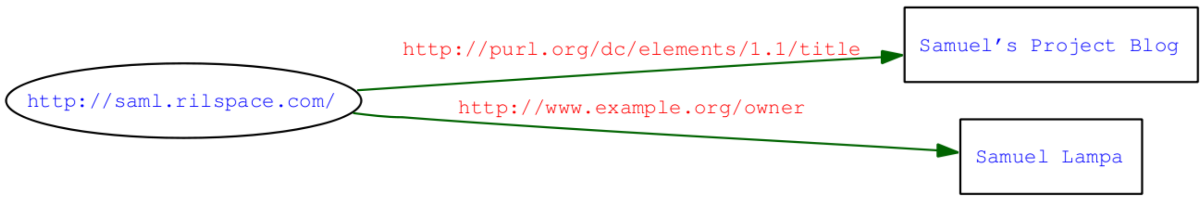

http://saml.rilspace.com/
http://purl.org/dc/elements/1.1/title
http://www.example.org/owner
Samuel's Project Blog
Samuel Lampa

Figure 1: A simple example of an RDF graph, in N-triples format above and graphically represented below.

Berners-Lee [15], inventor of the *World Wide Web* and currently the director of the World Wide Web Consortium (W3C) which is the organization that develops the standards and technologies for the semantic web, among many other things.

The semantic web technologies are not the only ones that have been addressing the data integration challenges in the life sciences, but they have been described as the currently most promising ones [6, 8, 16] and the only approach able to integrate large amounts of highly heterogeneous data with a reasonable effort, thus making it appropriate for widespread use [17, 18]. With this background, it might not come as a surprise that the bioinformatics community has been early adopters of semantic web technologies [19] and that they are now finding their way into many life science fields [6]. The way semantic web technologies address data integration challenges will be discussed in more detail in subsubsection 2.2.1 below.

2.2.1 RDF

At the core of the semantic web is a technology called RDF, "Resource Description Framework" [20] which is a system for expressing knowledge about things, or *resources* by identifying each resource with a URI or "Unique Resource Identifier" and by defining relationships between resources and/or explicit data values. Relationships are described by triplets of URIs where the middle URI functions as a *predicate*, defining the type of relationship between the other two URIs, which are referred to as *subject* and *object* respectively. By connecting together resources, an RDF *graph* is formed, which is a network of interconnected nodes, represented by the URIs.

Since predicates are also identified with unique identifiers, they can be standardized and linked to a definition of a specified meaning. Such definitions are typically expressed in *ontologies*, or *vocabularies*. It is recommended to use already existing, standardized predicates as much as possible, as this will increase interoperability of data.

A notable feature of the triple model is that it removes the need for a separate schema for each data source by instead including the typing information in the data itself, something which has large consequences for data integration, by taking away the need for time

```
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix ex: <http://www.example.org/> .

<http://saml.rilspace.com/>     dc:title "Samuel's Project Blog";
      ex:owner "Samuel Lampa"
```

Figure 2: The RDF in Figure 1 shown in Notation 3 format.

consuming cooperation between data providers in order to enable large scale automation of data integration [2].

The main format for storing RDF is and XML-format called RDF/XML. RDF/XML is not very easy to type or read though, and is not intended as a format for manipulating directly as a notation format. Instead, it is used as stable storage format by software, which can take advantage of the XML representation by using existing XML parsing libraries for managing the data.

For the purpose of manipulating RDF directly, there exists multiple notation formats. One of the most widely used one is Notation 3 [21] (see Figure 2). Other common formats are N-Triples (see Figure 1) and Turtle [22] where N-triples is a subset of Turtle and Turtle is a subset of Notation 3 [23].

2.2.2 RDF Schema (RDFS)

While RDF provides a system for defining properties and relationships between entities, it provides no mechanism for *describing* these properties and relations [24]. In order to describe the meaning of properties and relations, vocabularies have to be introduced that connect properties and relations with a defined meaning. It is not realistic though to collect all possible semantics in one single vocabulary. Instead, the W3C provides a "language for expressing vocabularies" that allows the creation of specialized vocabularies and ontologies within all kinds of fields as complements to more general standardized vocabularies. This language for describing vocabularies is called RDFS or "Resource Description Framework Schema" [24] and contains predicates such as *subClassOf*, *type*, and *isDefinedBy* which can be used to define vocabularies.

We have mentioned *ontologies*, and RDFS can also be seen as a minimal *ontology* description language [25]. It serves partly the same role as ontology languages such as OWL (described in the next section) while not being as comprehensive. The terms "ontology" and "vocabulary" have some overlap in their use [26].

2.2.3 Web Ontology Language (OWL)

For the construction of advanced ontologies, e.g. for precisely defining the meaning of medical terms and how they relate to each other, the semantics of RDFS is not enough. In fact, the expressivity of RDF and RDF Schema is deliberately very limited [27]. Instead OWL, or "Web Ontology Language", was created for defining ontologies of arbitrary complexity.

```
@prefix : <http://www.example.com>
@prefix people: <http://www.example.com/people#>

SELECT ?grandson WHERE {
  people:grandpa :hasSon ?son .
  ?son :hasSon ?grandson .
}
```

Figure 3: SPARQL example of a query for the grandson of a person. The query will look up any item that has the *hasSon* relationship with grandpa. Then it will consequently look up the item that has the *hasSon* relationship to *that* person. This last one will then be the grandson of *grandpa*.

OWL has powerful features for expressing semantics by the use of logical operations such as *union*, *disjointness* etc.

OWL is available in three versions; OWL-Lite, OWL-DL and OWL-Full, with increasing degree of expressivity. This is because with increasing expressivity it is harder to create tools that provide full support for a language [27]. OWL-Lite is created to enable most tool-makers to fully support at least a basic level of OWL, while OWL-DL is the language with enough expressivity for most ontology building tasks. DL stands for "Description Logics". The most expressivity is provided by OWL-Full but it is assumed that no tool will be able to completely support OWL-Full [27].

2.2.4 SPARQL

SPARQL [28] is the W3C recommended language for querying of RDF data, or "RDF graphs". It is much like SQL, the query language for relational databases, but for querying RDF graphs instead of database structures. In SPARQL, one expresses an RDF graph pattern which includes variables that should be bound, indicated with a question mark, "?". A query for the grandson of a person could look like the SPARQL code shown in Figure 3. This code shows how one can select for a *grandson* even though there is no explicit *hasSon* predicate, by instead constructing a query that uses the *hasSon* two times in combination.

2.2.5 Reasoning

The semantic web way of storing knowledge, enables letting computers do more high-level knowledge processing tasks In addition to allowing integration and data querying, especially when using an expressive language such as OWL. One example is drawing conclusions that are not explicitly stated as facts but require several pieces of knowledge to be combined i.e. making *implicit* knowledge *explicit* [29].

A typical example where there exists *implicit* knowledge that could made *explicit* is the following three facts: *(i) Isac is the son of Abraham*, *(ii) Jacob is the son of Isac* and *(iii) the son of someones son is that persons grandson.* An *implicit* fact based on this knowledge, is that Jacob is the grandson of Abraham, and this is something that a computer program, or so called *reasoner*, can infer. This example is similar to the SPARQL query in Figure 3, with the difference that the SPARQL query does not produce a new RDF triple, it only returns the item that conforms to the query pattern. The process of performing this kind of higher-level knowledge processing is often called “reasoning” and software that performs it are referred to as “reasoners”.

There are different approaches to reasoning and the expression of knowledge that can be used for reasoning. The main approach for reasoning within the W3C recommended standards is so called “Description Logics” supported by OWL-DL, which provides means to describe relationships between classes and instances, by using *logical constructors*. Examples of typical such constructors are *disjointness*, *cardinality* and *transitivity*. An OWL-DL reasoner typically allows operations such as *validation*, which is about checking that all expressed knowledge in a knowledge base are logically consistent with each other, *classification*, which is to assign a class membership to an entity, based on a class description expressed in OWL, and finally *realization*, which is to create instances, based on a class description. [30]

2.2.6 Rule languages

Not everything is possible [31] or easy [25] to represent using description logics and OWL-DL though. Because of this there are also other approaches to inference and reasoning, most notably different rule-based formalisms. Rule-based systems have long been appreciated to be the best available means for representing the knowledge of human experts [32].

Rule-based formalisms typically work by specifying a pattern of facts and/or relations that should be satisfied in order for a certain so called “goal” statement to be true. Rules are in other words a way to express *conditional knowledge*.

Since there are already a number of different rule-based formalisms, including logic programming languages such as Prolog, which will be discussed below, higher-order logics, production systems etc [25], W3C has chosen not to endorse any specific rules language but instead to provide the “Rule Interchange Format” (RIF), developed by the RIF working group [33]. Some W3C members are also developing SWRL [34], which is a rule language compatible with the other W3C recommended standards for Semantic Web data, having the status of a “member submission”, i.e. it is not endorsed by W3C.

2.3 Tools for managing semantic data

This section will give the background for the scope of the current study, which is to study a Prolog-based tool for dealing with semantic web data and compare it to other, non-Prolog-based tools available in the Bioclipse workbench. This overview will cover comparisons of the tools as independent applications, since no previous study have compared them as integrated into Bioclipse.

2.3.1 Importance of performance of tools in Bioclipse

This study focuses on the performance of studied tools. Performance is of special importance in Bioclipse since it is focused on interactive use in contrast to some other related software such as Taverna, which is more focused on the creation of automated workflows. When using tools in an interactive fashion, response times are important. For example there are three typical time limits known [35] that has to be considered for avoiding different types of distractions in the workflow of a user: 0.1 seconds for making the user feel that things are happening instantaneously, 1 second for not interrupting the user's flow of thought, and 10 seconds for keeping the user's attention to a dialog. Depending on the type of task, one of these limits might be important to keep. An example policy could be that, updating information about the currently selected item in a list while scrolling, should not take more than 0.1 seconds, the most common data retrievement tasks should not take more than 1 seconds, and no query that is not seen as a batch operation that is not expected to be monitored by the user, should exceed 10 seconds etc [35].

2.3.2 Non-Prolog-based tools

The Semantic web field is maturing and mature tools, both proprietary and open-source, are starting to be available. Among reasoners, the most widely used ones include FaCT, FaCT++ [36], Pellet [30], Racer [37], RacerPRO [38], KAON2 [39] and Vampire [40]. Among more general RDF tools, the Jena RDF framework [41, 42] is a popular one, built in Java and open source.

An open source software for the life sciences with support for semantic web formats is the Bioclipse workbench. The aim of Bioclipse is integration of data and tools in a desktop application that is easy to use for scientists without extensive computer knowledge. Bioclipse supports the semantic web formats recommended by W3C by incorporating two mature and popular tools with somewhat complementary functionality; Jena and Pellet. Jena [41, 42] implements an RDF API, ability to read and write RDF in RDF/XML, Notation 3 and N-Triples formats and a SPARQL query engine [41] as well as RDF data stores, both for in-memory and disk-based storage. Pellet is a reasoner for the OWL-DL language [30] that can operate on RDF stores created in Jena. Pellet supports SPARQL querying and even though it is focused against OWL-DL queries it supports general RDF querying too via Jena. These tools, integrated into Bioclipse, have been applied to use cases in the life sciences before, for the purpose of demonstration [43].

A key difference of these semantic web tools as compared to Prolog-based ones, is that, by reflecting W3Cs partitioning of data formats into separate ones for storage, querying and inference, these tools are also almost always separated into modules directed at one of these formats at a time.

2.3.3 Prolog

Prolog is a popular general purpose logic programming language first released in 1972 [44], which uses rules as a means of expressing conditional knowledge. A typical Prolog rule,

```
 1 % == Some facts ==
 2 has_h_bond_donors( substanceX, 3 ).
 3 has_h_bond_donors( substanceY, 5 ).
 4 has_h_bond_donors( substanceZ, 7 ).
 5
 6 has_h_bond_acceptors( substanceX, 7 ).
 7 has_h_bond_acceptors( substanceY, 10 ).
 8 has_h_bond_acceptors( substanceZ, 13 ).
 9
10 has_molecular_weight( substanceX, 320 ).
11 has_molecular_weight( substanceY, 500 ).
12 has_molecular_weight( substanceZ, 500 ).
13
14 % == A Rule ("Rule of five" ala Prolog) ==
15 is_drug_like( Substance ) :-
16   has_h_bond_donors( Substance, HBDonors ),
17   HBDonors <= 5,
18   has_h_bond_acceptors( Substance, HBAcceptors ),
19   HBAcceptors <= 10,
20   has_molecular_weight( Substance, MW ),
21  MW < 500.
```

Figure 4: Example Prolog knowledge base, containing a number of facts (lines 2-12) and a rule (lines 15-21), consisting of a *head* (line 15), including the implication marker (':-') and a *body* (line 16-12). Variables start with a capital letter, while atoms ("constants") start with a lower-case letter. Commas denote conjunction (logical *AND*). Lines starting with a '%' character are comments.

like the one in Figure 4, consists of a head and a body, of which the semantics can be expressed as "if [body], then [head]", or equivalently; "To show [head], show [body]". The body can consist of multiple predicates which can be either simple facts, or refer to still other rules. The statements in the body can be combined together with comma characters which means logical "AND". Such a Prolog rule can be queried from inside Prolog in the way demonstrated in Figure 5.

Prolog allows carrying out RDF storage, querying and inference all within the same environment, which is not typically the case for conventional semantic web tools, where the storage and querying are often carried out in separate modules or applications. This unified environment in Prolog enables a more efficient execution of many tasks in Prolog than with conventional programming tools [45] by taking advantage of the reduced number of technological layers.

This study is focused on the performance for executing typical semantic data queries. Prolog is interesting in this regard, since, compared with traditional software, Prolog and prolog-based software have some uniquely attractive features that are very interesting when dealing with semantic data, and which have consequences for performance. Firstly, through the *semweb* package of the SWI-Prolog Prolog distribution, semantic data can

```
1 % == Querying the Rule ==
2 ?- is_drug_like( substanceX )
3 true.
4 ?- is_drug_like( X )
5 X = substanceX ;
6 X = substanceY.
```

Figure 5: Example of querying the Prolog rule in Figure 4, by sending an existing atom as input parameter (line 2), and by sending a variable (denoted with a capital first letter) as input parameter (line 4), which returns two results that satisfy the pattern (lines 5 and 6). Lines starting with a '%' character are comments.

be dealt with directly at the syntax of the Prolog language [46]. This is in contrast to conventional languages such as Java, where one has to explicitly program algorithms that can deal with the semantics – there is no way to deal with it directly in the Java language itself. This of course requires much more development time if starting from scratch with no previous Java code. And even if using preexisting Java code software than can deal with semantic data, the additional layer (of algorithms) *above* the Java language itself, can reasonably be anticipated to result in more complexity and thus slower programs in general. This is an underlying hypothesis in this study, which concentrates mainly on the execution speed of Prolog-based versus non-Prolog-based software, when applied to the same data and the same task.

The main disadvantages of Prolog and Prolog-based software, have been identified as the lack of easy-to-use resources for dealing with web protocols and documents and the limited availability of skilled Prolog programmers [46]. The former has been addressed, with the development of the semweb package for SWI-Prolog.

It should be mentioned that Prolog offers some unique characteristics compared to non-Prolog-based tools, that are important to take into account when using them. One such is the *closed world assumption*, as opposed to the *open world assumption* taken by most non-Prolog-based semantic web tools. A tool taking the closed world assumption basically assumes that it has all knowledge, with the consequence that it if can not prove the validity of a certain fact, it concludes that this fact is wrong. Under the *open world assumption*, it would instead only have concluded that it does not know whether the fact is true or wrong [30]. This is a problem for implementing semantic web applications in Prolog, since it effectively causes incompatibility with other semantic web tools, which support the W3C format/technology stack, which take the *open world assumption* [47].

A search on Google scholar [48] reveals numerous applications of life science use cases for Prolog. There are, however, not many reported that take the semantic web approach. One example is a Prolog-based toolkit developed for bioinformatics tasks [49], named BLIPKIT for "Biological LogIc Programming toolKIT". Blipkit builds upon the *semweb* package in SWI-Prolog and is available at http://www.blipkit.org.

2.4 Research question and project aims

Based on the limited experience of using Prolog for semantic web based approaches in the life sciences, and with using Prolog as an integrated part of a workbench system like Bioclipse in particular, it was decided to integrate SWI-Prolog into Bioclipse and evaluate it against life science use cases comparing it with the already available semantic web tools in Bioclipse. Furthermore, since most previous Prolog/semantic web use cases focus on discrete types of data, while the life sciences often uses highly quantitative, numerical data, the use case was chosen as to contain numerical data.

The research question for the project is formulated as follows:

"How do biochemical questions formulated as Prolog queries compare to other solutions available in Bioclipse in terms of speed?"

3 Methods

In this section, the methods used in this study to answer the research question will be presented. The methods include open standards, open source software and open data, as well as the procedure used in the performance tests in the study.

3.1 Used Open Standards

The main semantic web standard used was *RDF*, "Resource Description Framework" [20], which is the foundational language for data and knowledge on the semantic web, provided and recommended by the World Wide Web Consortium (W3C). Therefore, all data used in this study was made available in the RDF format, specifically in RDF/XML, which is an XML serialization of RDF graphs.

A second standard used is *SPARQL*, the W3C recommended language for querying of RDF data, or "RDF graphs". It has many similarities with SQL, while it describes RDF graph patterns instead of database schema patterns, and uses common bounding variables instead of the `join` keyword in SQL, for merging of sub graphs/tables. SPARQL is supported by Jena and Pellet, and was used for RDF querying in these tools in this study [28].

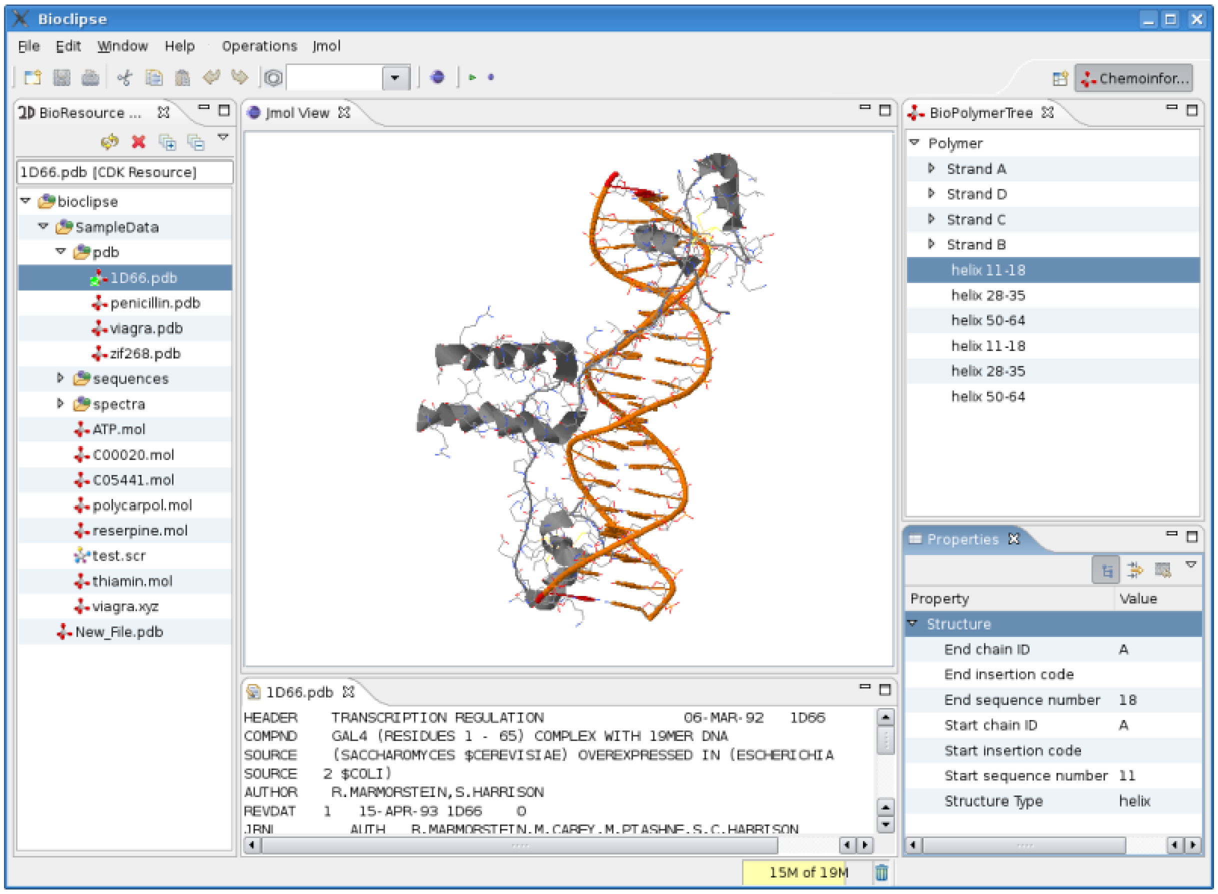
3.2 Used Open Source software

Bioclipse [50] is a free and Open Source workbench for the life sciences, and was used as the framework in which the evaluated tools were integrated and the querying experiments performed. The Bioclipse version used in this study was `2.2.0.RC2`.

Bioclipse is based on Eclipse [51] which is mostly known as a software development platform, but today it is a highly modular framework that can be extended in many directions, and is today used by many software vendors as a foundation for derivative end products and software development platforms. On top of Eclipses modular framework Bioclipse adds a modular system for integration of existing software through so called managers, which expose the software's functionality into the graphical user interface (GUI), as demonstrated by the screen-shot in Figure 6. Bioclipse additionally provides a Javascript based scripting environment into which managers expose functionality [52]. The scripting environment was utilized for the software evaluations in this study.

Bioclipse is designed for easy integration of existing software. Some kinds of integration, such as via command line is trivial while more programmatic integration can be eased with the help of a recently developed plug-in SDK (Software Development Kit) wizard, which guides the developer through a set of choices and automatically creates all the required file structure to let the developer start adding code to a plug-in [53]. The SDK was used for creating the SWI-Prolog integration plug-in used in this study.

Jena [41,42] is a semantic web framework for Java that was already available in Bioclipse where it accounts for most of the RDF related functionality. Since Jena is an established and readily available RDF framework developed in a conventional programming language,



```
HEADER    TRANSCRIPTION REGULATION                06-MAR-92   1D66
COMPND    GAL4 (RESIDUES 1 - 65) COMPLEX WITH 19MER DNA
SOURCE    (SACCHAROMYCES $CEREVISIAE) OVEREXPRESSED IN (ESCHERICHIA
SOURCE   2 $COLI)
AUTHOR    R.MARMORSTEIN,S.HARRISON
REVDAT   1   15-APR-93 1D66    0
JRNL        AUTH   R.MARMORSTEIN,M.CAREY,M.PTASHNE,S.C.HARRISON
```

Property	Value
Structure	
End chain ID	A
End insertion code	
End sequence number	18
Start chain ID	A
Start insertion code	
Start sequence number	11
Structure Type	helix

Figure 6: A Screenshot of the Bioclipse workbench. Here the JMol 3D structure viewer is open in the graphical user interface and used to visualize the 3D structure of a piece of DNA (the double-helical formed structure) and a DNA-binding protein.

it was chosen for evaluation in this study against the technically quite different SWI-Prolog, for querying RDF data. The version used was `2.6.2`.

Jena provides an RDF API, ability to read and write RDF in RDF/XML, Notation 3 and N-Triples formats and a SPARQL query engine [41]. Jena provides both one in-memory, and two types of persistent RDF storage. The SDB storage engine utilizes a mySQL database for the storage, while the TDB storage is a specialized store, and has in general better performance than SDB [54]. In this study, the in-memory store and the TDB disk-based store were used in the evaluations.

The Bioclipse manager methods used in this study, of which the functionality is provided by Jena, are the following:

- *rdf.createStore()* – Creates a new (in-memory) RDF store. This method was used to create in-memory RDF stores for performance testing Jena.

- *rdf.importFile(IRDFStore store, String target, String format)* – Loads an RDF file in the given content format ("RDF/XML", "N-TRIPLE", "TURTLE" and "Notation 3") into the given store. This method was used to load RDF in RDF/XML format into an RDF Store.

- *rdf.sparql(IRDFStore store, String query)* – Returns the results matching the SPARQL query. This method was used for executing the SPARQL code used for performance testing of Jena.

Pellet [30] is an Open Source reasoner for the OWL-DL language written in Java and available in Bioclipse prior to this study. Pellet can operate on RDF stores (in-memory or disk-based ones) of the Jena RDF framework. Pellet supports SPARQL querying, and even though it is focused against OWL-DL queries, it also supports general RDF querying via Jena. Therefore it was included in the RDF querying evaluations in this study.

The Bioclipse manager methods used in this study, of which the functionality is provided by Pellet, are the following:

- *pellet.createStore(String tripleStoreDirectoryPath)* – Creates a new scalable Pellet-targeted store, (using the Jena TDB package, which stores on disk as a complement to memory, for scalability). The *tripleStoreDirectoryPath* is the path (relative to the Bioclipse workspace) to a folder to use for the triple store. This method was used for creating the disk-based stores for use with Pellet and Jena.

- *pellet.reason(IRDFStore store, String query)* – Returns the results matching the SPARQL query using the Pellet reasoner. This method was used for executing SPARQL code for performance testing of Pellet.

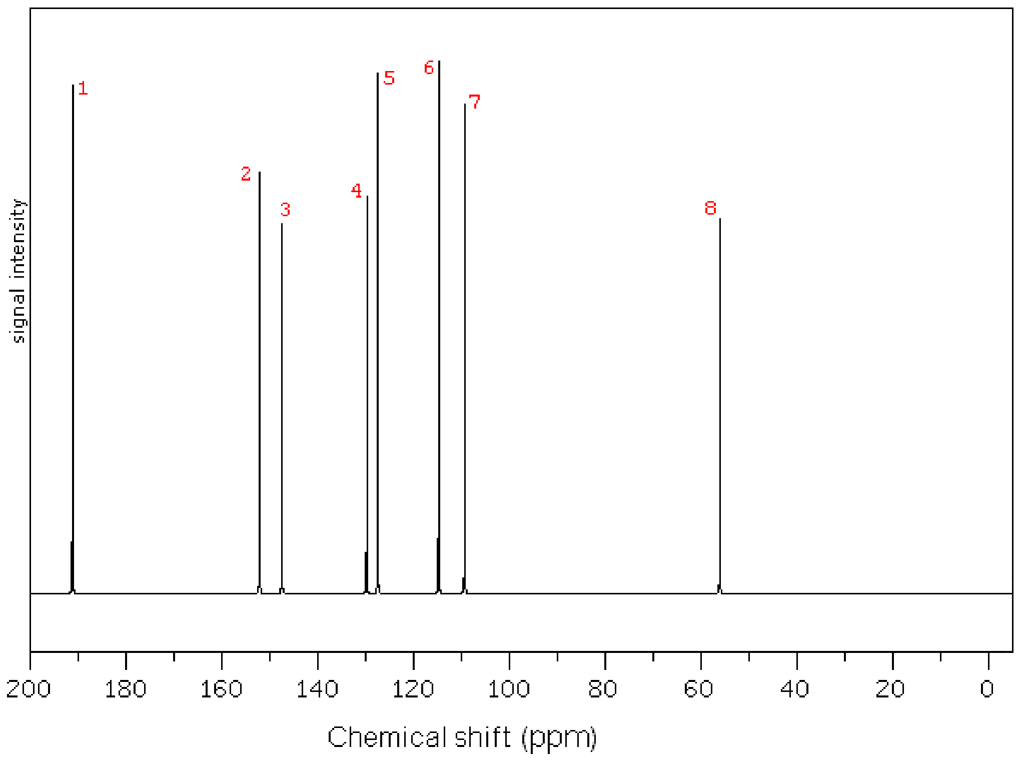
SWI-Prolog [55, 56] is an open source Prolog environment with mature support for handling RDF data at the triple level [57]. SWI-Prolog also provides a Java programming level API for easy integration of the Prolog engine in Java programs. The mature RDF support and the Java API made SWI-Prolog the Prolog environment of choice for integration in Bioclipse for evaluation against the already available RDF tools. The version of SWI-Prolog used in this study was `5.7.15`.

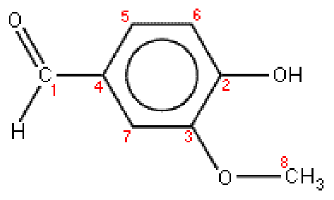
3.3 Used Open Data

NMRShiftDB [58, 59] is a free community built web database (available at `http://www.nmrshiftdb.org`) for organic chemical structures and their corresponding NMR data. As of September, 2009, NMRShiftDB contained 25712 searchable spectra.

An example ^{13}C NMR spectrum can be seen in Figure 7 which shows the ^{13}C NMR spectrum itself above and the corresponding molecule below, where each magnetically unique carbon atom in the molecule corresponds to one peak in the spectrum.

An RDF-ized version of the NMRShiftDB data was used in this study, but with empty NMR shift value nodes filled with zeros, to avoid errors in Jena, and by substituting a custom type definition, `nmr:ppm`, which was not properly recognized by Jena, with





ppm	Int.	Location
191.21	955	1
152.18	791	2
147.50	692	3
129.77	746	4
127.49	975	5
114.75	1000	6
109.34	920	7
56.10	706	8

Figure 7: An example of a ^{13}C NMR Spectrum, showing the corresponding molecule below. Each carbon atom in the molecule corresponds to one peak in the spectrum. Each peak has an intensity (peak height) and a shift value (horizontal placement), which is a relative value (relative to a standard sample) and is measured in ppm.

`http://www.w3.org/2001/XMLSchema#decimal`, taking into account also that prefixes do not work for data type definitions.

A histogram showing the distribution of shift values in the full data set can be seen in Figure 8. As can be seen, the largest values have few other values close to them.

The resulting data describes molecules connected to ^{13}C NMR spectra, which each contains a number of peaks, each of which has a shift value. Peaks in fact have both

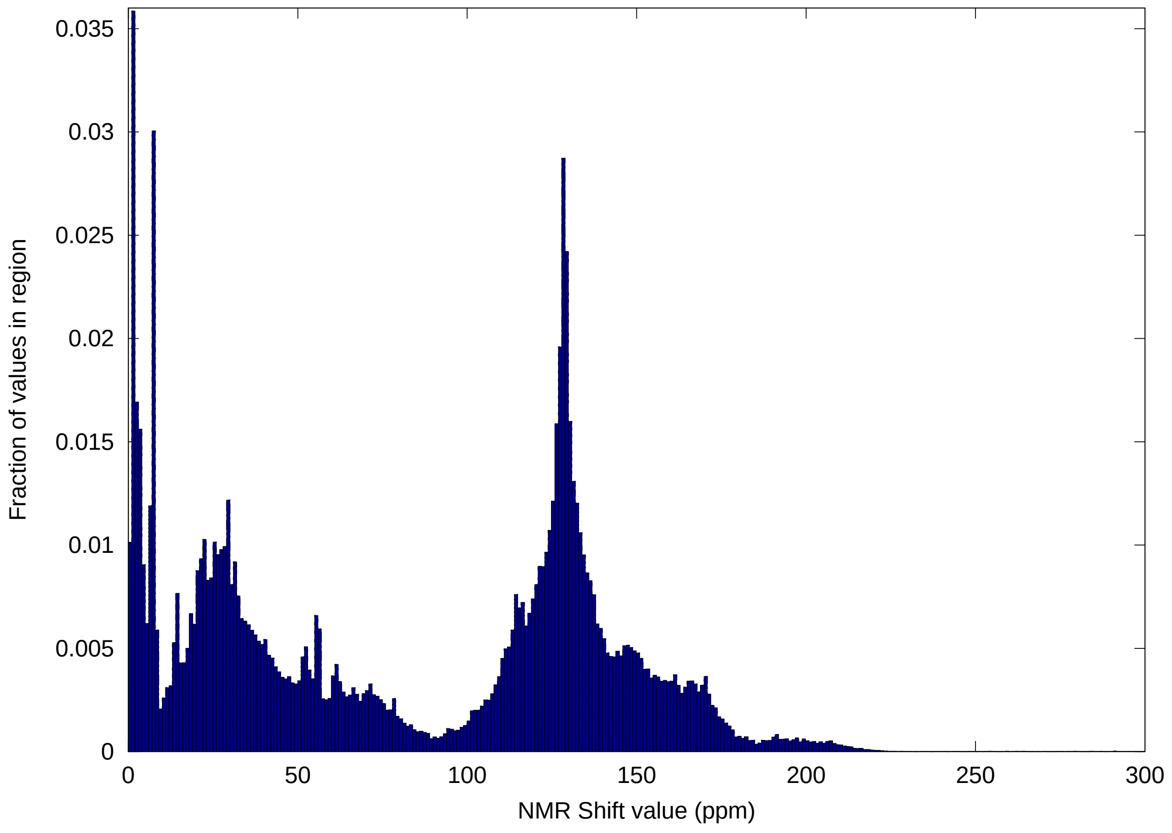


Figure 8: Normalized histogram of NMR peak shift values in the biggest data set, containing 25000 spectrum, or over 1 million peak shift values. The histogram shows only values up to 300 ppm, and excludes 75 peaks, with values between 300 and 578.3 ppm.

intensity (peak height) and shift (peak placement) but in this study we are only looking at the *shift* values it is common use for ^{13}C spectra to assume that the shift values contain enough information to discriminate between spectra, and therefore to omit intensity values [59]. Intensity could easily be included if wanted, in the same way as the shift values are included [60]. An extract of the final data, in Notation 3 format, can be seen in Figure 9 where a spectrum and three of its peaks with their corresponding shift and intensity values have been included for demonstration.

Five differently sized data sets were created by copying the data set into five copies and truncating to different sizes, containing 5000, 10000, 15000, 20000 and 25000 spectra each. The full RDF-ized dataset, which was using data retrieved from NMRShiftDB in September 2009, contained 25712 spectra, but it was chosen to stop at 25000 to get a consistent distance between the data points.

```
@prefix    : <http://www.nmrshiftdb.org/onto#> .
        xsd: <http://www.w3.org/2001/XMLSchema#> .

    <http://pele.farmbio.uu.se/nmrshiftdb/?moleculeId=234>
         :hasSpectrum <http://pele.farmbio.uu.se/nmrshiftdb/?spectrumId=4735>;
         :moleculeId  "234".

    <http://pele.farmbio.uu.se/nmrshiftdb/?spectrumId=4735>
         :hasPeak <http://pele.farmbio.uu.se/nmrshiftdb/?s4735p0>,
                  <http://pele.farmbio.uu.se/nmrshiftdb/?s4735p1>,
                  <http://pele.farmbio.uu.se/nmrshiftdb/?s4735p2>,

    <http://pele.farmbio.uu.se/nmrshiftdb/?s4735p0>     a :peak;
         :hasShift "17.6"^^xsd:decimal .
    <http://pele.farmbio.uu.se/nmrshiftdb/?s4735p1>     a :peak;
         :hasShift "18.3"^^xsd:decimal .
    <http://pele.farmbio.uu.se/nmrshiftdb/?s4735p2>     a :peak;
         :hasShift "22.6"^^xsd:decimal .
```

Figure 9: Example of RDF-ized data from NMRShiftDB, in Notation 3 format. The 'a' keyword is a shortcut for the 'rdf:type' predicate

3.4 Performance comparison

As a biochemical use case for comparing the performance of Jena vs Pellet vs SWI-Prolog we used the RDF–ized ^{13}C NMR Spectrum data from NMRShiftDB to do a simple "dereplication" of spectra as described in [59]. That is, given a reference spectrum, represented as a list of shift values, search for spectra with the same shifts but allowing variation within a limit. This limit was chosen to 3 ppm, as it has been reported that the uncertainty of shift values typically is within 2-3 ppm. [61].

The Prolog query for the use case can be seen as an *NMR Spectrum similarity search.* This search query was programmed in Prolog. The Prolog code for this can be found in Figure 17, and is also available as a Bioclipse script on myExperiment [62] as workflow 1271.

A SPARQL Query for the same task was also constructed, and can be found in Figure 16 and is also available as a Bioclipse scripts on myExperiment [62] as workflow 1268, 1269 and 1270, for Jena with in-memory RDF store, TDB store and for Pellet, respectively. A notable difference between the Prolog program and the SPARQL query is that while the Prolog program takes the list of shift values representing the reference spectrum *as a parameter*, the SPARQL query requires the values to be *hard coded* in the SPARQL query.

Finally another Prolog implementation of the query was made, that mimicked the structure of the above mentioned SPARQL query as closely as possible. For example, the values to search for were hard coded in the Prolog code instead of taken from an input variable. This Prolog query can be found in Figure 18, and is also available as a Bioclipse script on myExperiment [62] as workflow 1272.

For the performance comparison, the Prolog code in Figures 17 and Figure 18 were executed in SWI-Prolog, and the SPARQL query in Figure 16 was executed in Jena and Pellet. Each run was performed by scripting the loading of RDF data and measurement of execution time of the actual queries in Bioclipse scripts. The timing was done only for the querying phase, excluding the RDF loading phase, by storing and comparing timestamps in the Bioclipse scripting environment.

Three iterations were performed for each specific size of the data sets. Bioclipse was restarted after each run in order to avoid memory and/or caching related bias. The Java Virtual Machine start up parameters for memory usage were changed in the Eclipse `.product` file as follows.

- Maximum heap size, `-Xmx` was changed from `512M` to `1196M`.
- Maximum permanent memory used, `-XX:MaxPermSize=` was changed from `128M` to `512M`.

All performance tests were performed on an Asus UL30A laptop, with the following specifications:

- **CPU:** 1.3GHz Intel Core 2 Duo SU7300 (Dual core)
- **RAM:** 4 Gb, SO-DIMM DDR3
- **Hard drive:** 5400 RPM, 500 Gb
- **Operating system:** 32 bit Ubuntu 9.10 desktop edition.

During the tests, the CPU clocking frequency was locked to the maximum, 1.3 GHz, in Ubuntu's power management console, to avoid CPU speed changes due to power saving options.

3.5 MyExperiment website

The MyExperiment website [62] was used to publish Bioclipse scripts demonstrating the usage of the SWI-Prolog integration plug in, as well as for doing the performance comparisons. The uploaded scripts are listed below:

- NMR Spectrum similarity search with SWI-Prolog in Bioclipse (`http://www.myexperiment.org/workflows/1116`)
- NMR Spectrum similarity search benchmark for Jena with in-memory RDF store (`http://www.myexperiment.org/workflows/1268`)
- NMR Spectrum similarity search benchmark for Jena with TDB RDF store (`http://www.myexperiment.org/workflows/1269`)

- NMR Spectrum similarity search benchmark for Pellet with TDB RDF store (http://www.myexperiment.org/workflows/1270)
- NMR Spectrum similarity search benchmark for SWI-Prolog (http://www.myexperiment.org/workflows/1271)
- NMR Spectrum similarity search benchmark for SWI-Prolog, minimal version (http://www.myexperiment.org/workflows/1272)

4 Results

The research question for this study was *"How do biochemical questions formulated as Prolog queries compare to other solutions available in Bioclipse in terms of speed?"*. The comparison of a Prolog-based approach to querying semantic web data, against non-Prolog-based ones, required the integration of a Prolog environment into Bioclipse, which was therefore part of the results of the study. This section will therefore first present the integration of SWI-Prolog, together with some examples that demonstrate the way SWI-Prolog can be used for working with semantic web data in Bioclipse. To answer the research question, a performance comparison of SWI-Prolog against Jena and Pellet was also performed as described in the methods section, and the results from this comparison will also be presented here in this section.

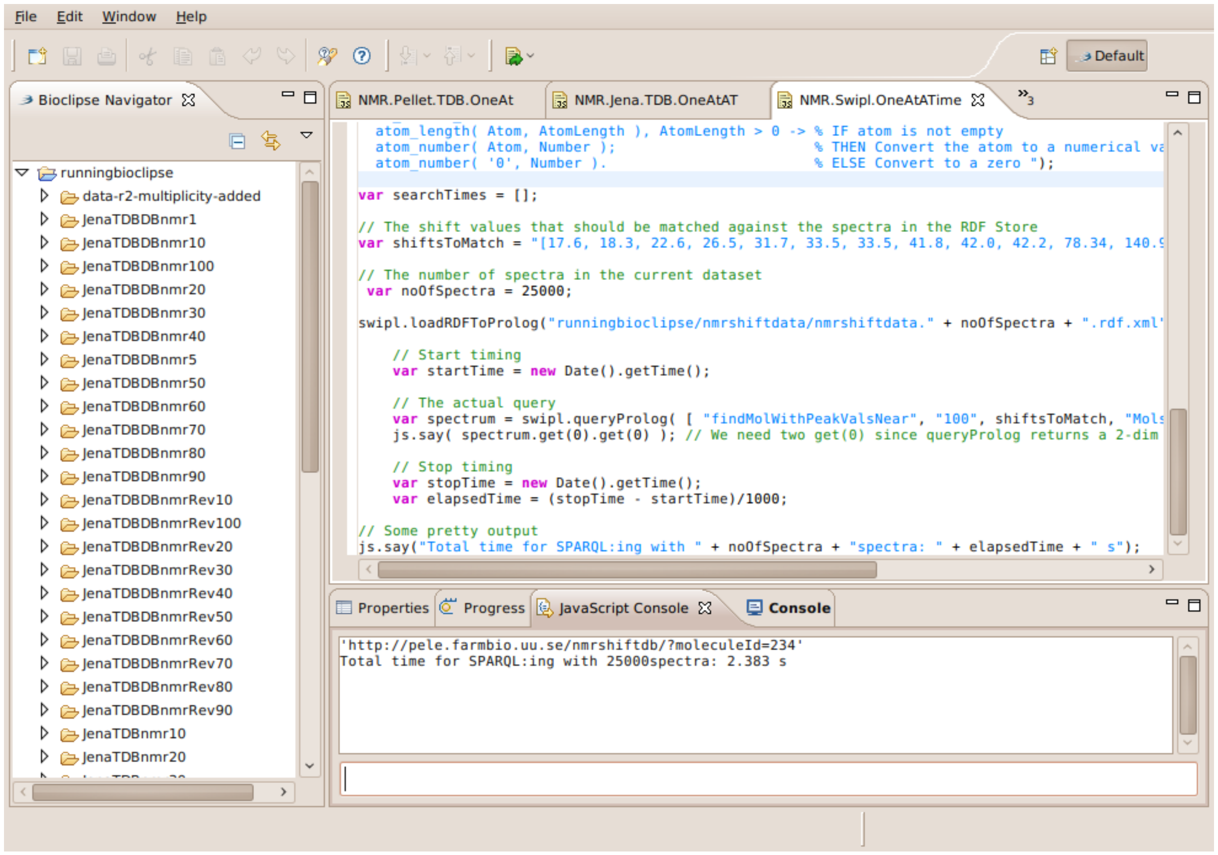
4.1 Implementation of research methods

In order to fairly compare SWI-Prolog with other RDF tools in Bioclipse, SWI-Prolog was integrated into Bioclipse. The integration constituted a considerable part of the project time. The integration was done using the JPL Java Prolog API which provides classes and methods for constructing Prolog atoms and queries, as well as a Prolog library which is used to connect to a Prolog engine [63].

Convenience methods were developed and made available to the scripting environment in Bioclipse, including methods for loading RDF data into Prolog, for loading Prolog code from inside the scripting environment, and for performing execution of query-like Prolog predicates that populate one or more variables and returns the result as an array. See Figure 10 for an example of the SWI-Prolog plug-in in action inside Bioclipse, just having returned the results of a query to the console in the bottom of the picture. A full list of

swipl.getActualArgs()	Prints the arguments that the current Prolog engine was called with. Used for error tracking
swipl.init()	Initialize the Prolog engine (loads SWI-Prolog modules) etc.
swipl.loadPrologCode(String prologCode)	Loads Prolog code, as stored in the prologCode variable, into the Prolog engine
swipl.loadPrologFile(String filepath)	Loads a file with Prolog code
swipl. loadRDFFromFile(String rdfFile)	Invokes loading of an rdfFile into Prolog. Makes use of SWI-Prologs semweb package
swipl.printLibPath()	Prints the value of java.library.path
swipl.queryProlog(String[] prologFunction resultLimit prologArguments)	Takes an array of strings; 1: The Prolog function, 2: Max no. of results, and the rest being arguments passed to the Prolog function
swipl.queryRDF(String Subject, String Prolog, String Object)	Executes a Prolog query and prints all solutions

Table 1: Bioclipse manager methods of the SWI-Prolog integration plug-in.



```
  atom_length( Atom, AtomLength ), AtomLength > 0 -> % IF atom is not empty
  atom_number( Atom, Number );                       % THEN Convert the atom to a numerical va
  atom_number( '0', Number ).                        % ELSE Convert to a zero ");

var searchTimes = [];

// The shift values that should be matched against the spectra in the RDF Store
var shiftsToMatch = "[17.6, 18.3, 22.6, 26.5, 31.7, 33.5, 33.5, 41.8, 42.0, 42.2, 78.34, 140.9

// The number of spectra in the current dataset
 var noOfSpectra = 25000;

swipl.loadRDFToProlog("runningbioclipse/nmrshiftdata/nmrshiftdata." + noOfSpectra + ".rdf.xml'

    // Start timing
    var startTime = new Date().getTime();

    // The actual query
    var spectrum = swipl.queryProlog( [ "findMolWithPeakValsNear", "100", shiftsToMatch, "Mols
    js.say( spectrum.get(0).get(0) ); // We need two get(0) since queryProlog returns a 2-dim

    // Stop timing
    var stopTime = new Date().getTime();
    var elapsedTime = (stopTime - startTime)/1000;

// Some pretty output
js.say("Total time for SPARQL:ing with " + noOfSpectra + "spectra: " + elapsedTime + " s");
```

```
'http://pele.farmbio.uu.se/nmrshiftdb/?moleculeId=234'
Total time for SPARQL:ing with 25000spectra: 2.383 s
```

Figure 10: A Screenshot of the Bioclipse workbench running the integrated SWI-Prolog engine. In the top right window, a Bioclipse script is open for editing, which contains Prolog code, and Javascript code for performing comparison tests of SWI-Prolog. In the lower right window, the result from running the query is shown; a spectrum URI, together with the timing results for the execution.

the manager methods in the SWI-Prolog integration plug-in can be found in Table 1.

The *swipl.loadRDFFromFile(rdfFile)* manager method allows loading of RDF data from a file in the Bioclipse workspace. The *swipl.loadPrologFile(prologFile)* manager method allows loading Prolog code from a file while *swipl.loadPrologCode(prologCode)* manager method allows loading from the scripting environment - that is, for example code stored in a variable. Thus one can store Prolog code along with the main Bioclipse Javascript code, enabling to keep a complete Bioclipse work flow definition in one single file. This way of storing Prolog code embedded in the scripting environment is demonstrated in the Bioclipse script in Figure 11. In this example, the Prolog code is loaded in two sessions. First, RDF name spaces are registered, then some RDF data is loaded into Prolog where after a convenience Prolog predicate is loaded, which is subsequently queried to retrieve RDF data for peaks with shifts equaling 42.2 ppm in this case.

```
// Namespaces have to be registered before loading RDF data
swipl.loadPrologCode("
:- rdf_register_ns(nmr, 'http://www.nmrshiftdb.org/onto#').
:- rdf_register_ns(xsd, 'http://www.w3.org/2001/XMLSchema#').       ");

// Load RDF Data
swipl.loadRDFToProlog("nmrshiftdata.rdf.xml");

// Load the Prolog method we want to use for querying
swipl.loadPrologCode("
hasSpectrumWithPeakWithShift( Molecule, Shift ) :-
  rdf( Molecule, nmr:hasSpectrum, Spectrum ),
  rdf( Spectrum, nmr:hasPeak, Peak ),
  rdf( Peak,     nmr:hasShift, literal(type(xsd:decimal, Shift)) ). ");

// Query and output all molecules with shift = '42.2', limiting to ten results
var molecules = swipl.queryProlog( [ "hasSpectrumWithPeakWithShift",
                                     "10", "Molecules", "'42.2'" ] );
// Output results to console
js.print(molecules);
```

Figure 11: Example Bioclipse script that demonstrates how Prolog code can be stored in Bioclipse script variables and loaded into Prolog using the *swipl.loadPrologCode(prologCode)* manager method. Furthermore, RDF data can be loaded using the *swipl.loadRDFToProlog(RDF datafile)* manager method. The Prolog convenience predicate *hasSpectrumWithPeakWithShift* is used to query for RDF data. In this case it queries for molecules with spectrum peak shifts equaling 42.2 ppm.

```
// Load RDF Data, and Prolog code
swipl.loadRDFToProlog("nmrshiftdata.rdf.xml");
swipl.loadPrologFile( "findMolWithPeakValsNear.pl" );

// Define the list of peak shifts that should be searched for
var shifts = "[ 17.6, 18.3, 22.6, 26.5, 31.7, 33.5, 33.5, 41.8 ]";

// Query the findMolWithPeakValsNear Prolog method, limiting to 10 results
var spectra = swipl.queryProlog( [ "findMolWithPeakValsNear", "10", shifts, "Mols" ] )

// Print out the results
js.print( spectra );
```

Figure 12: Bioclipse script that loads RDF data and a file with Prolog code (see Figure 18 for the Prolog code used), and performs a query for all molecules with shift peaks with near-matches of the provided list of shifts, using the *findMolWithPeakValsNear* Prolog predicate.

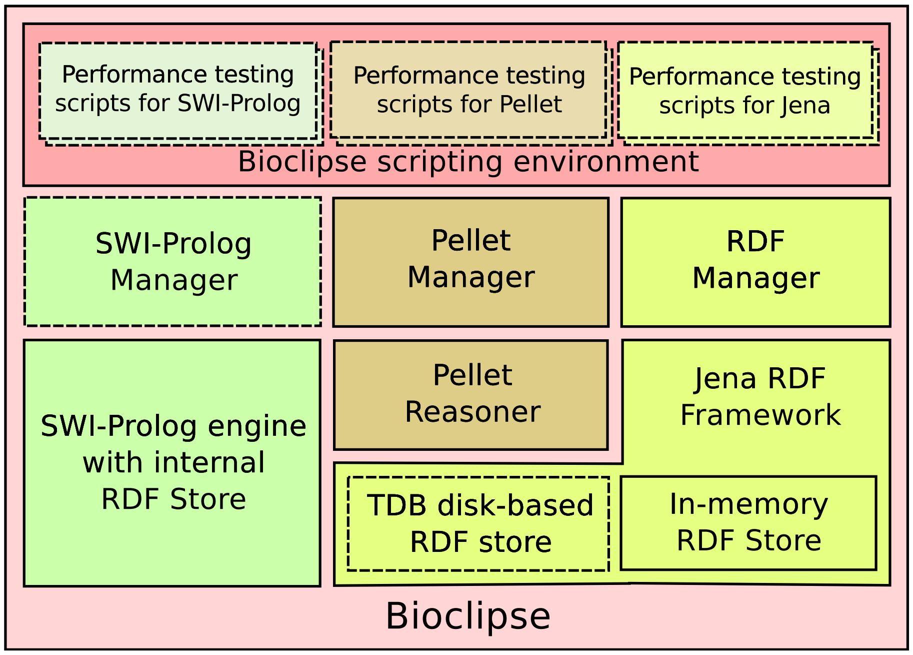

Performance testing scripts for SWI-Prolog
Performance testing scripts for Pellet
Performance testing scripts for Jena
Bioclipse scripting environment
SWI-Prolog Manager
Pellet Manager
RDF Manager
SWI-Prolog engine with internal RDF Store
Pellet Reasoner
Jena RDF Framework
TDB disk-based RDF store
In-memory RDF Store
Bioclipse

Figure 13: Using Bioclipse and SWI-Prolog, a scripting environment extension (manager) was developed to allow creation and execution of Bioclipse scripts for performance comparison of SWI-Prolog with Jena and Pellet. The functionality developed in this project is marked with dashed lines.

Figure 12 demonstrates how Prolog code can also be stored in a separate file in the Bioclipse workspace, and loaded with the *swipl.loadPrologFile(prologFile)* manager method. The actual Prolog code in this case is the same as is used in the performance evaluation in this study, and is available in Figure 18. Figure 13 shows the logical structure of the integrated SWI-Prolog engine in Bioclipse, related to the other semantic web tools, Jena and Pellet.

4.1.1 Availability and requirements

- **Project name:** SWI-Prolog integration plug-in for Bioclipse
- **Source code:** http://github.com/samuell/bioclipse.swipl
- **Operating system(s):** Ubuntu Linux
- **Programming language:** Java
- **Other requirement(s):** Java 1.6.0 or higher (http://java.sun.com), Bioclipse 2.2.0 or higher (http://www.bioclipse.net)
- **License:** GNU Library or Lesser General Public License (LGPL)

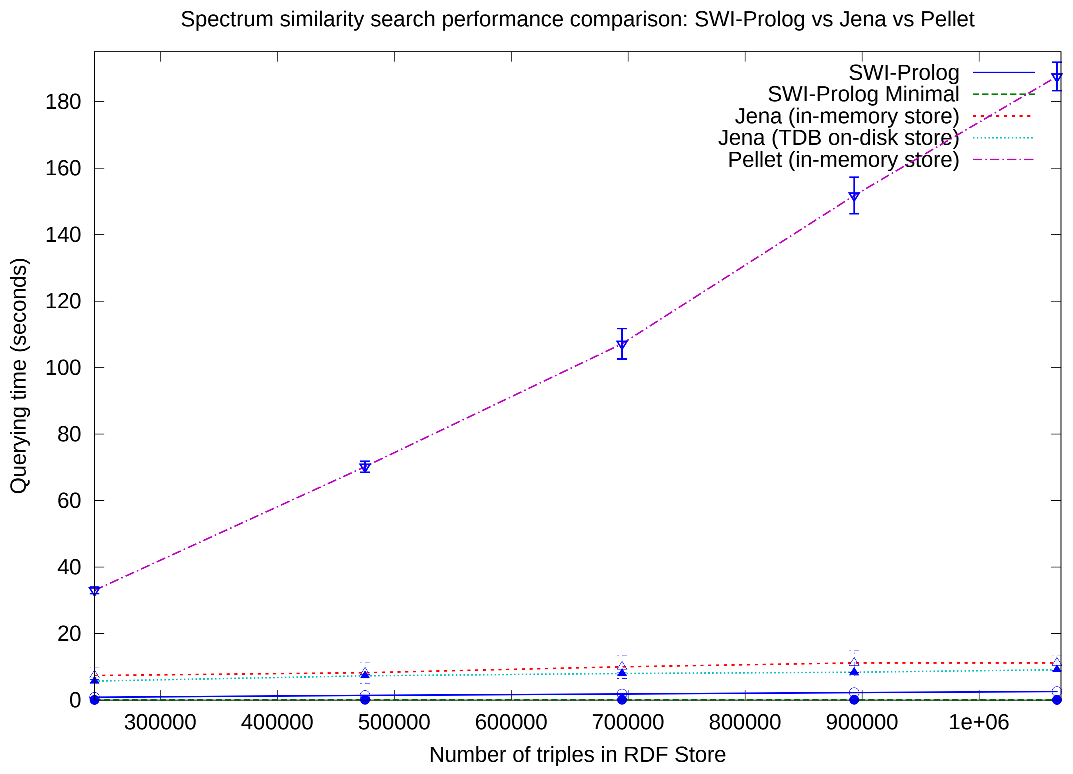


Figure 14: Comparison of execution of a spectrum similarity search query/program, between SWI-Prolog (SWIPL) for the longer (Figure 18) and the shorter (Figure 18) Prolog program, Jena with disk-based RDF store (TDB) and in-memory RDF store, and Pellet (with in-memory RDF store). The solid line shows the average of repeated tests while the error bars show the standard deviation.

4.2 Performance comparison: SWI-Prolog vs Jena vs Pellet

The execution time of the SPARQL query in Figure 16 in Pellet and Jena, and the Prolog queries in Figures 17 and 18 in SWI-Prolog, were measured according to the description in the methods section. The test includes Jena with the in-memory RDF Store as well as the TDB disk-based one while the combination Pellet/TDB store had to be excluded since it did not complete for more than an hour even for the smallest data set with 5000 spectra, making performance comparisons practically unfeasible.

The results are shown in Figures 14 and 15. In Figure 14, Pellet is included while it is omitted in Figure 15 in order to get a zoom in on the Jena/SWI-Prolog difference. Average and standard deviation were calculated and can be seen in Figures 14 and 15, where the solid line plots the average and the error bars shows the standard deviation.

As can be seen, the longer Prolog program is around three times as fast as Jena with disk-based RDF store, and around six times faster than Jena with in-memory store. The minimal, SPARQL like Prolog query in turn is almost ten times faster than even the other

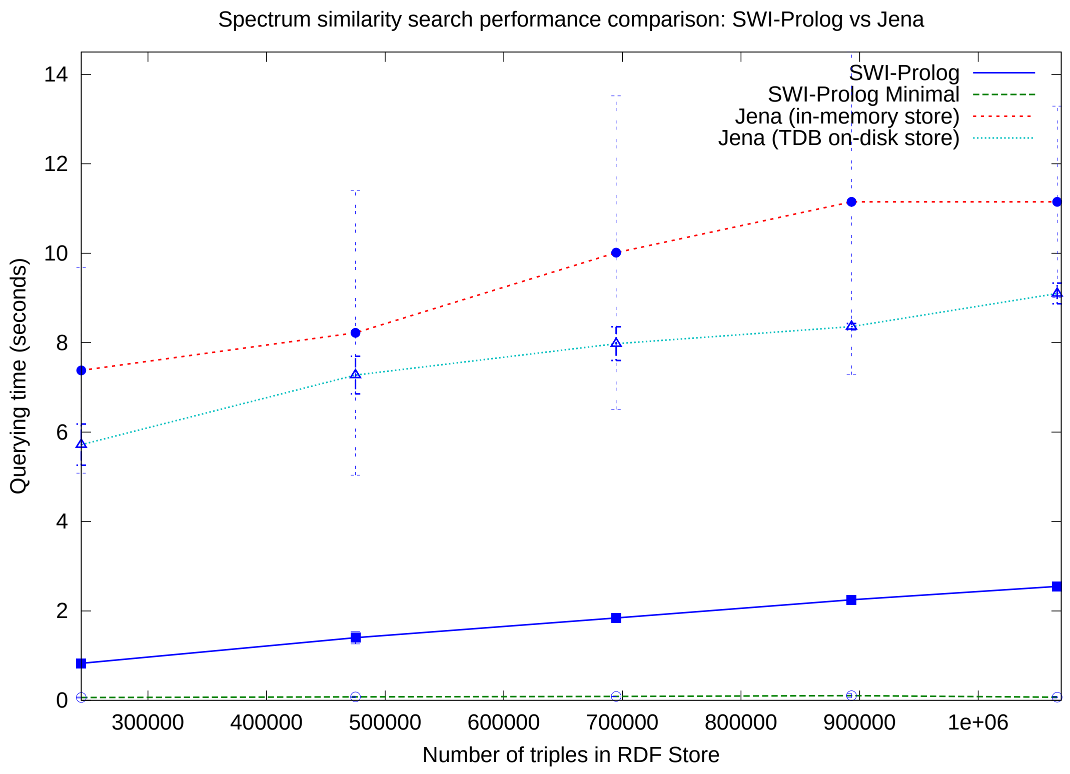


Figure 15: Comparison of execution of a spectrum similarity search query/program, between SWI-Prolog (SWIPL) for the longer (Figure 18) and the shorter (Figure 18) Prolog program, Jena with disk-based RDF store (TDB) and in-memory RDF store. Pellet is excluded here in order to zoom in on the other results. The solid line shows the average of repeated tests while the error bars show the standard deviation.

Prolog program. As can be seen, all queries scale approximately linearly.

It should be noted that in the SPARQL like Prolog query, the order of the values are reversed so that larger shift values are searched for first, followed by smaller and smaller shift values. This is an optimization heuristic that takes into account the fact that the largest shift values are much less common, and therefore less prone to overlap with other values, than lower shift values. This can be seen in the histogram in Figure 8. When executing the query with the values sorted from small to large, the querying was so slow that it could for practical reason not be fully performed (results not shown). Even on the smallest data set, the query had not finished after more than one hour, so it was excluded from further tests.

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX afn: <http://jena.hpl.hp.com/ARQ/function#>
PREFIX fn: <http://www.w3.org/2005/xpath-functions#>
PREFIX nmr: <http://www.nmrshiftdb.org/onto#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT  ?m
WHERE {
  ?m nmr:hasSpectrum ?s .
  ?s nmr:hasPeak [ nmr:hasShift ?s1 ] ,
                 [ nmr:hasShift ?s2 ] ,
                 [ nmr:hasShift ?s3 ] ,
                 [ nmr:hasShift ?s4 ] ,
                 [ nmr:hasShift ?s5 ] ,
                 [ nmr:hasShift ?s6 ] ,
                 [ nmr:hasShift ?s7 ] ,
                 [ nmr:hasShift ?s8 ] ,
                 [ nmr:hasShift ?s9 ] ,
                 [ nmr:hasShift ?s10 ] ,
                 [ nmr:hasShift ?s11 ] ,
                 [ nmr:hasShift ?s12 ] ,
                 [ nmr:hasShift ?s13 ] ,
                 [ nmr:hasShift ?s14 ] ,
                 [ nmr:hasShift ?s15 ] ,
                 [ nmr:hasShift ?s16 ] .
FILTER ( fn:abs(?s1 - 203.0) < 3 ) .
FILTER ( fn:abs(?s2 - 193.4) < 3 ) .
FILTER ( fn:abs(?s3 - 158.3) < 3 ) .
FILTER ( fn:abs(?s4 - 140.99) < 3 ) .
FILTER ( fn:abs(?s5 - 78.34) < 3 ) .
FILTER ( fn:abs(?s6 - 42.2) < 3 ) .
FILTER ( fn:abs(?s7 - 42.0) < 3 ) .
FILTER ( fn:abs(?s8 - 41.8) < 3 ) .
FILTER ( fn:abs(?s9 - 33.5) < 3 ) .
FILTER ( fn:abs(?s10 - 33.5) < 3 ) .
FILTER ( fn:abs(?s11 - 31.7) < 3 ) .
FILTER ( fn:abs(?s12 - 26.5) < 3 ) .
FILTER ( fn:abs(?s13 - 22.6) < 3 ) .
FILTER ( fn:abs(?s14 - 18.3) < 3 ) .
FILTER ( fn:abs(?s15 - 17.6) < 3 ) .
FILTER ( fn:abs(?s16 - 0) < 3 ) .
}
```

Figure 16: SPARQL query used in Jena and Pellet in the performance comparison between Pellet, Jena and SWI-Prolog. The query is available as a Bioclipse script on MyExperiment [62] as workflow 1268 for Jena with in-memory store, 1269 for Jena with TDB store and 1270 for Pellet with TDB store.

```
% Register RDF namespaces, for use in the convenience methods at the end
:- rdf_register_ns(nmr, 'http://www.nmrshiftdb.org/onto#').
:- rdf_register_ns(xsd, 'http://www.w3.org/2001/XMLSchema#').

findMolWithPeakValsNear( SearchShiftVals, Mols ) :-
  % Pick the Moleculess in 'Mol', that match the pattern:
  %% listPeakShiftsOfMol( Mol, MolShiftVals ),
  %% containsListElemsNear( SearchShiftVals, MolShiftVals )
  % ... and collect them in 'Mols'.
  % A Mol(ecule)s shift values are collected and compared against the given
  % SearchShiftVals. Then, in 'Mols', all 'Mol's, for which their shift values
  % match the SearchShiftVals, are collected.
  setof( Mol,
         ( listPeakShiftsOfMol( Mol, MolShiftVals ),
           containsListElemsNear( SearchShiftVals, MolShiftVals )),
         [Mols|MolTail] ).

% Given a 'Mol', give its shiftvalues in list form, in 'ListOfPeaks'
listPeakShiftsOfMol( Mol, ListOfPeaks ) :-
  hasSpectrum( Mol, Spectrum ),
  findall( ShiftVal,
           ( hasPeak( Spectrum, Peak ),
           hasShiftVal( Peak, ShiftVal ) ),
  ListOfPeaks ).

% Compare two lists to see if list2 has near-matches for each value in list1
containsListElemsNear( [ElemHead|ElemTail], List ) :-
  memberCloseTo( ElemHead, List ),
  ( containsListElemsNear( ElemTail, List );
  ElemTail == [] ).

%%% Recursive construct: %%%
% Test first the end criterion:
memberCloseTo( X, [ Y | Tail ] ) :-
  closeTo( X, Y ).
% but if the above doesn't validate, do recursively continue with the tail of List2:
memberCloseTo( X, [ Y | Tail ] ) :-
  memberCloseTo( X, Tail ).
% Numerical near-match
closeTo( Val1, Val2 ) :-
  abs(Val1 - Val2) =< 3.

%%% Convenience accessory methods %%%
hasShiftVal( Peak, ShiftVal ) :-
  rdf( Peak, nmr:hasShift, literal(type(xsd:decimal, ShiftValLiteral))),
  atom_number_create( ShiftValLiteral, ShiftVal ).
hasSpectrum( Subject, Predicate ) :-
  rdf( Subject, nmr:hasSpectrum, Predicate).
hasPeak( Subject, Predicate ) :-
  rdf( Subject, nmr:hasPeak, Predicate).

% Wrapper method for atom_number/2 which converts atoms (string constants)
% to numbers. Avoids exceptions on empty atoms, instead converting to a zero.
atom_number_create( Atom, Number ) :-
  atom_length( Atom, AtomLength ), AtomLength > 0 -> % IF atom is not empty
  atom_number( Atom, Number );                       % THEN Convert to a num. val.
  atom_number( '0', Number ).                        % ELSE Convert to a zero
```

Figure 17: Prolog script used in the performance comparison between Jena, Pellet and SWI-Prolog. This code, together with the code in Figure 18 was used for execution in SWI-Prolog, while Figure 16 shows the SPARQL code that was used in Jena and Pellet. The code is available as a Bioclipse script on MyExperiment [62] as workflow 1271.

```
:- rdf_register_ns(nmr, 'http://www.nmrshiftdb.org/onto#').
:- rdf_register_ns(xsd, 'http://www.w3.org/2001/XMLSchema#').

select_mol_w_pshifts( Mol ) :-
  q( Mol, 203.0 ),
  q( Mol, 193.4 ),
  q( Mol, 158.3 ),
  q( Mol, 140.99 ),
  q( Mol, 78.34 ),
  q( Mol, 42.2 ),
  q( Mol, 42.0 ),
  q( Mol, 41.8 ),
  q( Mol, 33.5 ),
  q( Mol, 33.5 ),
  q( Mol, 31.7 ),
  q( Mol, 26.5 ),
  q( Mol, 22.6 ),
  q( Mol, 18.3 ),
  q( Mol, 17.6 ),
  q( Mol, 0 ).

%%% Query method %%%
q( Mol, RefShiftVal ) :-
  rdf( Mol, nmr:hasSpectrum, Spec),
  rdf( Spec, nmr:hasPeak, Peak),
  rdf( Peak, nmr:hasShift, literal(type(xsd:decimal, ShiftValLiteral))),
  atom_number_fixzero( ShiftValLiteral, ShiftVal ),
  abs(ShiftVal - RefShiftVal) =< 3.

atom_number_fixzero( Atom, Num ) :-
  atom_length( Atom, AtomLen ), AtomLen > 0 -> % IF atom is not empty
  atom_number( Atom, Num );                    % THEN Convert to num. val.
  atom_number( '0', Num ).                     % ELSE Convert to a zero
```

Figure 18: Minimal Prolog query, constructed to minimize the amounts of characters used, and to mimic the structure of the SPARQL query in Figure 16. This code, together with the code in Figure 17 was used for execution in SWI-Prolog. The code is available as a Bioclipse script on MyExperiment [62] as workflow 1272.

5 Discussion

The research question for this study was *"How do biochemical questions formulated as Prolog queries compare to other solutions available in Bioclipse in terms of speed?"*. The comparison of a Prolog-based approach to querying semantic web data, against non-Prolog-based ones, required first the integration of a Prolog environment into Bioclipse, followed by a performance comparison of SWI-Prolog versus the other semantic web tools in Bioclipse. The work done in this project has thus provided both new functionality, and new information on the performance of this functionality.

5.1 Integration of SWI-Prolog in Bioclipse

The integration of SWI-Prolog into Bioclipse has given access to the increased performance and convenience for working with at least some types of data by taking advantage of the Prolog-based semantic web functionality in the *semweb* package of SWI-Prolog.

The ability to store data processing workflows inside a single Bioclipse script file facilitates sharing of workflows between scientists, and makes it possible to use the full functionality of the Prolog programming language for defining data processing workflows in Bioclipse.

5.2 Performance comparison

The results show that querying RDF is faster in Prolog than in non-Prolog-based software for a simply structured knowledge base. This is what could be expected based on earlier studies [39] which showed that a datalog[1] reasoner is faster for simple structured ontologies with large number of assertions, while it was not faster for data with much more hierarchically structured ontologies, i.e. much more branched ones. It also gives support to the underlying hypothesis of this study, as presented in the introduction, that the additional layers needed in conventional programming languages as compared to Prolog, tend to create slower programs in general.

It is an interesting observation that writing the Prolog query on the simpler form (Figure 18) made it amenable to heuristic optimization by sorting the values searched for, while this was not possible in the longer Prolog program (Figure 17). The possibility for heuristics based optimization based on statistics of the data set would be an interesting area for further research. Indeed, such studies have been done for the TDB disk-based store bundled with Jena [64].

One possible explanation why the shorter code was amenable to the mentioned optimization and not the longer, is that the shorter query makes better use of Prologs backtracking mechanism by allowing to skip value comparisons for spectra where the first value did not find a match anyway. The longer Prolog querys list operations might rule out the ability to stop the value testing for a spectra when it encounters a mis-match. This hypothesis needs further verification though.

[1]Datalog is approximately a subset of Prolog in functionality

A surprising result is that Jena is around double as fast with the TDB, disk-based, store than with the in-memory store. It is probable though, that the disk-based store is not in fact accessing the disk at all, for this data set. The fact that we could load the data set in the in-memory store hints that TDB is also only storing the data in memory since the disk is only used when data does not fit into memory, according to [65]. We can thus draw no conclusions on why TDB is faster than the in-memory store, and the exact explanation requires further research. It would also be an area of further research to see how TDB scales with the data set size compared to the in memory store for data sets which do not fit into memory.

Otherwise, it is encouraging to see that all tools scale (approximately) linearly with the size of the data sets, and that the querying time can be argued to be within practical limits for everyday use after optimization by ordering of queried values was done.

To get a general picture though, further studies would have to evaluate performance and expressivity also for data sets with different characteristics. Furthermore, more research on the problems of the closed/open world assumption issue and the resulting incompatibility problems is needed before the use of Prolog for semantic web applications in the life sciences can be advocated.

5.3 Convenience of the Prolog language

As can be seen in Figures 16, 17 and 18 it is possible to express the same thing with approximately the same amount of code in SPARQL as in Prolog. The Prolog query in Figure 18 does however not require rewriting any part of the actual query in order to query for another set of peak shift values. In the SPARQL query, one has to add things to the code at three places in order to add another shift value, and even if the *number* of values remains constant one would have to do as many changes as there are values. The Prolog query in Figure 17, on the other hand, can take the list of values as an input parameter, which makes it much easier to use programmatically from Bioclipse scripts. It should be noted though, that the SPARQL query can also be generated with Bioclipse scripts, though that is not a feature of SPARQL itself.

When the query is written as in the Prolog query in Figure 18, then, while he number of places where one needs to update the query in order to change shift values is similar to the SPARQL query, then still, changing the limit value (which is 3 in this study), is much easier in the Prolog queries, since it requires changing only one occurrence of the value as compared to 16 occurrences in the SPARQL query. The reason for this clearly comes from the ability in Prolog to create functions. This enabled to break out repeated operations into a separate function (the `q` function in Figure 18). The same feature is fundamental for being able to use Prolog in the programmatic way as demonstrated by the longer Prolog program in Figure 18.

5.4 Implications for data integration

While the research question of this study focused on the comparison of Prolog-based versus non-Prolog-based semantic web technologies, the context of the study was data integration challenges in the life sciences and the meeting of these challenges using semantic web technologies. The current study does not provide a comprehensive study on how the new Prolog-based functionality in Bioclipse can be used to address data integration challenges, but it has shown that Prolog-based tools can successfully be used within Bioclipse and provide increased performance and convenience for at least some kinds of data. This suggests Prolog as interesting for application in general data integration scenarios, especially for data with many numerical values, and where performance is an important issue.

5.5 Limitations of SWI-Prolog compared with Jena and Pellet

In the background, the lack of skilled Prolog programmers was mentioned as a general drawback for the Prolog language. It is also the authors' impression, that the Prolog language will probably take more time to learn than the SPARQL query language, used in Jena and Pellet, because of SPARQL's close similarity to the database query language SQL, which is a very common language. Additionally, a drawback of SWI-Prolog specifically, against Jena and Pellet, is that since it is not written in Java, it is not as portable (i.e. the same code can not easily be executed) to different platforms such as Mac, Windows, Linux etc. Instead the source code has to be compiled separately for each platform. This also has the result that the SWI-Prolog Bioclipse integration plugin will not be as *portable* as Bioclipse itself.

5.6 Using Bioclipse for increased reproducibility

The choice of doing the software evaluations in Bioclipse allowed to store the complete experiment as a script that can easily be shared with other scientists, so that others can easily repeat what has been done. The scripts used for performance testing in this study are available at the MyExperiment website [62] and are listed in subsection 3.5.

6 Conclusion

The research question of this study was *"How do biochemical questions formulated as Prolog queries compare to other solutions available in Bioclipse in terms of speed?"*. To answer this question, SWI-Prolog was integrated into Bioclipse. As we have demonstrated, SWI-Prolog has provided increased performance and convenience for working with the specific type of data used in this study, compared to the other semantic web tools available in Bioclipse. The increased performance was demonstrated by drastically shorter execution times. An example of the increased convenience of the Prolog language, compared with SPARQL, is how the Prolog language allows to break out repetitive statements into functions.

Prolog has some drawbacks, like that it is likely to take longer time to learn than SPARQL, and the limited portability of the SWI-Prolog distribution, as it is not programmed in Java.

The good performance of SWI-Prolog and the convenience of the Prolog language hints at a general applicability in data integration and we suggest more research into the implications of the closed/open world assumptions issue as well as more performance evaluations of SWI-Prolog, with other types of data.

We also suggest further research to see how the TDB RDF store in Jena, scales with respect to the in-memory store, for data sets that do not fit into memory.

7 Acknowledgments

I have a number of people to thank for making this project possible and/or a more pleasant experience. First I want to thank professor Jarl Wikberg for the possibility to do the project in his group at the Pharmaceutical Biosciences department. I want to thank the Bioclipse team, Ola, Carl, Jonathan, Arvid and Annzi and Egon, for a very good time in the group, all the nice sushi lunches, IRC conversations and not the least all the helpful assistance in getting me going with Bioclipse development. I have really enjoyed the time in the group! I want to thank Brandon Ibach and Evren Parsia on the pellet-users mailing list for very helpful discussions on SPARQL intricacies. I want to thank my scientific reviewer Mats Gustafsson for accepting to review the project, my opponents, and last but not least my supervisor, Egon Willighagen for all the support, good advice and encouragement throughout the project. It has really given a lot of motivation and I have learnt a lot of practical wisdom in the craft of doing research that will surely follow me in my future career.

References

[1] Stephan Philippi and Jacob Köhler. Addressing the problems with life-science databases for traditional uses and systems biology. *Nature reviews. Genetics*, 7(6):482–8, June 2006.

[2] Lincoln D Stein. Towards a cyberinfrastructure for the biological sciences: progress, visions and challenges. *Nature reviews. Genetics*, 9(9):678–88, September 2008.

[3] Donna Maglott, Jim Ostell, Kim D Pruitt, and Tatiana Tatusova. Entrez Gene: gene-centered information at NCBI. *Nucleic acids research*, 35(Database issue):D26–31, January 2007.

[4] T Hubbard, D Barker, E Birney, G Cameron, Y Chen, L Clark, T Cox, J Cuff, V Curwen, T Down, R Durbin, E Eyras, J Gilbert, M Hammond, L Huminiecki, a Kasprzyk, H Lehvaslaiho, P Lijnzaad, C Melsopp, E Mongin, R Pettett, M Pocock, S Potter, a Rust, E Schmidt, S Searle, G Slater, J Smith, W Spooner, a Stabenau, J Stalker, E Stupka, a Ureta-Vidal, I Vastrik, and M Clamp. The Ensembl genome database project. *Nucleic acids research*, 30(1):38–41, January 2002.

[5] UCSC Genome browser – http://genome.ucsc.edu/. Online, retreived June 1, 2010.

[6] Erick Antezana, Martin Kuiper, and Vladimir Mironov. Biological knowledge management: the emerging role of the Semantic Web technologies. *Briefings in bioinformatics*, 10(4):392–407, 2009.

[7] David B Searls. Data integration: challenges for drug discovery. *Nature reviews. Drug discovery*, 4(1):45–58, 2005.

[8] Carole Goble and Robert Stevens. State of the nation in data integration for bioinformatics. *Journal of Biomedical Informatics*, 41:687–693, 2008.

[9] Kei-Hoi Cheung, H Robert Frost, M Scott Marshall, Eric Prud'hommeaux, Matthias Samwald, Jun Zhao, and Adrian Paschke. A journey to Semantic Web query federation in the life sciences. *BMC bioinformatics*, 10 Suppl 10:S10, January 2009.

[10] Robert Hoffmann. A wiki for the life sciences where authorship matters. *Nature genetics*, 40(9):1047–51, September 2008.

[11] Alvis Brazma. On the Importance of Standardisation in Life Sciences. *Bioinformatics*, 17(2):113–114, February 2001.

[12] Barend Mons, Michael Ashburner, Christine Chichester, Erik van Mulligen, Marc Weeber, Johan den Dunnen, Gert-Jan van Ommen, Mark Musen, Matthew Cockerill, Henning Hermjakob, Albert Mons, Abel Packer, Roberto Pacheco, Suzanna Lewis, Alfred Berkeley, William Melton, Nickolas Barris, Jimmy Wales, Gerard Meijssen, Erik Moeller, Peter Jan Roes, Katy Borner, and Amos Bairoch. Calling on a million minds for community annotation in WikiProteins. *Genome biology*, 9(5):R89, 2008.

[13] Maurizio Lenzerini. Data Integration: A Theoretical Perspective. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 233–246, Madison, 2002.

[14] Alan Ruttenberg, Jonathan A Rees, Matthias Samwald, and M Scott Marshall. Life sciences on the Semantic Web: the Neurocommons and beyond. *Briefings in bioinformatics*, 10(2):193–204, March 2009.

[15] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, pages 37–43, 2001.

[16] Alan Ruttenberg, Tim Clark, William Bug, Matthias Samwald, Olivier Bodenreider, Helen Chen, Donald Doherty, Kerstin Forsberg, Yong Gao, Vipul Kashyap, June Kinoshita, Joanne Luciano, M Scott Marshall, Chimezie Ogbuji, Jonathan Rees, Susie Stephens, Gwendolyn T Wong, Elizabeth

Wu, Davide Zaccagnini, Tonya Hongsermeier, Eric Neumann, Ivan Herman, and Kei-Hoi Cheung. Advancing translational research with the Semantic Web. *BMC Bioinformatics*, 8 Suppl 3:S2, January 2007.

[17] Erick Antezana, Ward Blondé, Mikel Egaña, Alistair Rutherford, Robert Stevens, Bernard De Baets, Vladimir Mironov, and Martin Kuiper. BioGateway: a semantic systems biology tool for the life sciences. *BMC bioinformatics*, 10 Suppl 10:S11, 2009.

[18] Kei-Hoi Cheung, Kevin Y Yip, Andrew Smith, Remko Deknikker, Andy Masiar, and Mark Gerstein. YeastHub: a semantic web use case for integrating data in the life sciences domain. *Bioinformatics (Oxford, England)*, 21 Suppl 1:i85–96, June 2005.

[19] Robert Stevens, Olivier Bodenreider, and Yves a. Lussier. Semantic Webs for Life Sciences: Session Introduction. *Pacific Symposium on Biocomputing 2006*, 115:112–115, 2006.

[20] Frank Manola and Eric Miller. RDF Primer, [http://www.w3.org/TR/rdf-primer/]. Online, Retrieved April 2010.

[21] Tim Berners-Lee (ed.). Notation 3 - a readable language for data on the web [http://www.w3.org/DesignIssues/Notation3.html], May 2010. Retrieved 1 May 2010.

[22] David Beckett and Tim Berners-Lee. Turtle - Terse RDF Triple Language [http://www.w3.org/TeamSubmission/turtle/]. [Online; accessed 3 May 2010].

[23] Wikipedia. N-triples — wikipedia, the free encyclopedia. Online, 2 May 2010.

[24] Dan Brickley and R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema [http://www.w3.org/TR/rdf-schema/]. Online, Retrieved April 2010.

[25] Nigel Shadbolt, Wendy Hall, and Tim Berners-Lee. The Semantic Web Revisited. *IEEE Intelligent Systems*, 2006.

[26] Wikipedia. Ontology (information science) — wikipedia, the free encyclopedia. Online, 3 May 2010.

[27] Grigoris Antoniou and Frank Van Harmelen. *Web Ontology Language: OWL*, pages 67–92. Springer, Berlin; New York, 2004.

[28] Grant Clark Kendall (ed.). Sparql protocol for rdf [http://www.w3.org/TR/2005/WD-rdf-sparql-protocol-20050527/], April 2010.

[29] Franz Baader, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook - Theory, Implementation and Applications.* Cambridge University Press, Cambridge, 2003.

[30] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical OWL-DL reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2):51–53, June 2007.

[31] Ian Horrocks, Peter F Patel-Schneider, Sean Bechhofer, and Dmitry Tsarkov. OWL rules: A proposal and prototype implementation. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(1):23–40, 2005.

[32] Frederick Hayes-Roth. Rule-based systems. *Communications of the ACM*, 28(9):921–932, September 1985.

[33] W3C. RIF Working Group, [http://www.w3.org/2005/rules/]. Online, Retrieved April 2010.

[34] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML, [http://www.w3.org/Submission/SWRL/], 2004. Online, Retrieved April 2010.

[35] Jakob Nielsen. *Usability Engineering (Interactive Technologies).* Morgan Kaufmann, 1 edition, September 1993.

[36] Dmitry Tsarkov and Ian Horrocks. FaCT++ Description Logic Reasoner: System Description. In U. Furbach and N. Shankar, editors, *IJCAR 2006*, pages 292–297, Berlin, Heidelberg, 2006. Springer-Verlag.

[37] Volker Haarslev and Ralf Möller. RACER System Description. In R. Goré, A. Leitsch, and T. Nipkow, editors, *IJCAR 2001*, pages 701–705, Berlin, Heidelberg, 2001. Springer-Verlag.

[38] RacerPRO website [http://www.racer-systems.com/]. Online, 30 April 2010.

[39] Boris Motik and Ulrike Sattler. A Comparison of Reasoning Techniques for Querying Large Description Logic ABoxes. In M. Hermann and A. Voronkov, editors, *LPAR 2006*, pages 227–241, Berlin, Heidelberg, 2006. Springer-Verlag.

[40] Alexandre Riazanov and Andrei Voronkov. The design and implementation of V AMPIRE. *Ai Communications*, 15:91–110, 2002.

[41] Jena semantic web framework for java [http://openjena.org/]. Online, April 2010.

[42] Brian Mcbride. Jena: A Semantic Web Toolkit. *IEEE Internet Computing*, (NOVEMBER/DECEMBER):55–59, 2002.

[43] Egon L Willighagen and Jarl E S Wikberg. Linking Open Drug Data to Cheminformatics and Proteochemometrics. In M. S. Marshall, A. Burger, P. Romano, A. Paschke, and A. Splendiani, editors, *Semantic Web Applications and Tools for Life Sciences*, 2010.

[44] Robert A Kowalski. The Early Years of Logic Programming. *Communications of the ACM*, 31(I), 1988.

[45] Gergely Lukácsy, Péter Szeredi, and Balázs Kádár. *Prolog Based Description Logic Reasoning*, volume 5366 of *Lecture Notes in Computer Science*, pages 455–469. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[46] Jan Wielemaker, Michiel Hildebrand, and Jacco Van Ossenbruggen. Using Prolog as the fundament for applications on the Semantic Seb. In *CEUR-WS*, pages 1–16, 2007.

[47] Ian Horrocks, Bijan Parsia, Peter Patel-schneider, and James Hendler. Semantic Web Architecture: Stack or Two Towers? *Order A Journal On The Theory Of Ordered Sets And Its Applications.*

[48] Google Scholar [http://scholar.google.com/]. Online, May 2010.

[49] Christopher J Mungall. *Experiences Using Logic Programming in Bioinformatics*, volume 5649 of *Lecture Notes in Computer Science*, pages 1–21. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[50] Ola Spjuth, Tobias Helmus, Egon L Willighagen, Stefan Kuhn, Martin Eklund, Johannes Wagener, Peter Murray-Rust, Christoph Steinbeck, and Jarl E S Wikberg. Bioclipse: an open source workbench for chemo- and bioinformatics. *BMC bioinformatics*, 8(1):59, 2007.

[51] Eclipse software development platform [http://eclipse.org/]. Online, April 2010.

[52] Ola Spjuth, Jonathan Alvarsson, Arvid Berg, Martin Eklund, Stefan Kuhn, Carl Mäsak, Gilleain Torrance, Johannes Wagener, Egon L Willighagen, Christoph Steinbeck, and Jarl E S Wikberg. Bioclipse 2: a scriptable integration platform for the life sciences. *BMC bioinformatics*, 10:397, 2009.

[53] Bioclipse Software Development Kit [http://chem-bla-ics.blogspot.com/2009/08/making-bioclipse-development-easier-new.html], 2009.

[54] Pros and Cons for different Jena Backends — Semantic Overflow [http://www.semanticoverflow.com/questions/506/pros-and-cons-for-different-jena-backends-sdb-vs-tdb]. Online, April 2010.

[55] Report CW371. In Fred Mesnard and Alexander Serebrenik, editors, *Proceedings of the 13th International Workshop on Logic Programming Environments*, number November. Department of Computer Science, K.U.Leuven, 2003.

[56] SWI-Prolog [http://www.swi-prolog.org/]. Online, April 2010.

[57] Jan Wielemaker, Guus Schreiber, and Bob Wielinga. Prolog-Based Infrastructure for RDF: Scalability and Performance. In D. Fensel, editor, *ISWC 2003*, pages 644–658, Berlin, Heidelberg, 2003. Springer-Verlag.

[58] Christoph Steinbeck, Stefan Krause, and Stefan Kuhn. NMRShiftDB–Constructing a Free Chemical Information System with Open-Source Components. *Journal of Chemical Information and Modeling*, 43(6):1733–1739, November 2003.

[59] Christoph Steinbeck and Stefan Kuhn. NMRShiftDB – compound identification and structure elucidation support through a free community-built web database. *Phytochemistry*, 65(19):2711–7, 2004.

[60] Egon Willighagen, NMRShiftDB contributor (Personal communication, May 2010).

[61] Athanasios Tsipouras, John Ondeyka, Claude Dufresne, Seok Lee, Gino Salituro, Nancy Tsou, Michael Goetz, Sheo Bux Singh, and Simon K Kearsley. Using similarity searches over databases of estimated 13C NMR spectra for structure identification of natural product compounds. *Analytica Chimica Acta*, 316(2):161–171, November 1995.

[62] My experiment website [http://www.myexperiment.org/]. Online, April 2010.

[63] JPL: A bidirectional Prolog/Java interface [http://www.swi-prolog.org/packages/jpl/]. Online, May 2010.

[64] Andy Seaborne. TDB Optimizer, [http://openjena.org/wiki/TDB/Optimizer]. Online, May 2010.

[65] Andy Seaborne. TDB Architecture, [http://openjena.org/wiki/TDB/Architecture], 2009. Online, Retrieved April 2010.