5

Semantic Infrastructures

Jiří Dokulil, Jakub Yaghob and Filip Zavoral
Charles University in Prague
Czech Republic

1. Introduction

There is a well defined infrastructure (set of protocols and their implementations) for the current Internet. For example HTTP implemented by the Apache or IIS servers and in all internet browsers (clients). Or HTML, CSS and JavaScript for content presentation. While there is still room for improvement in standards conformance, the system works quite well.
The Semantic Web is not such case. Although there are many protocols and standards for data description, querying, semantic services etc., most of them have very few implementations, typically only prototypes. Many systems are developed without regard for interoperability, often as monolithic software.
But over the time, some prominent RDF handling systems have become increasingly popular, thus defining relatively stable and well accepted interfaces – both APIs and higher level interfaces. There are more such systems and there is no or limited interoperability among the systems, but thanks to their relatively wide adoption (compared to the state several years ago), they greatly improve the chance that two independently developed pieces of semantic web software can be made to work together.
Furthermore there are attempts by W3C to standardize some of the many interfaces such system requires to work. The most important is SPARQL query language together with SPARQL query protocol (Prud'hommeaux & Seaborne 2008). These standardize the whole process of making data queries. The standardized language is rather simple and the protocol may not be the most efficient one, but it still is a significant step towards a working, unified infrastructure.

2. Covered Systems

There is an abundance of existing systems supporting semantic web and semantization - a comprehensive list is maintained by W3C Consortium (Alexander 2009). Most of them are academic or scientific prototypes, many of which are unfortunately no longer supported. In this survey, we have included four most significant (in our opinion) systems that are still maintained and supported. In section 7, we also shortly mention some other systems.

2.1 Sesame

Sesame (Broekstra et al. 2002) is an open source RDF framework with support for RDF Schema inferencing and querying. Originally, it was developed as a research prototype for the EU research project On-To-Knowledge. Now, it is further developed and maintained by Aduna in cooperation with a number of volunteer developers. Sesame is one of the oldest systems, a lot of semantic projects used this infrastructure for storing their data. Currently, version 2.2.x is available, Sesame 3.0 with significant improvements in both API and implementation is pre-released.

2.2 Jena

Jena (Carrol et al., 2004) is a Java framework for building Semantic Web applications. It provides a programmatic environment for RDF, RDFS and OWL, SPARQL and includes a rule-based inference engine. Jena is open source and grown out of work with the HP Labs Semantic Web Programme.

2.3 Mulgara

The Mulgara Semantic Store is an Open Source, scalable, transaction-safe, purpose-built database for the storage and retrieval of metadata. It is an active fork of the Kowari (Wood 2005) project.

2.4 Redland

Redland (Beckett 2002) is a set of C libraries that provide support for RDF.

3. Architecture and Interfaces

In this section we explore the architecture and most important interfaces, both application and human-oriented.

3.1 Sesame

Sesame can be deployed on top of a variety of storage systems (relational databases, in-memory, filesystems, keyword indexers, etc.), and offers tools to developers to leverage the power of RDF and RDF Schema, such as a flexible access API, which supports both local and remote (through HTTP or RMI) access, and several query languages.

Figure 1 depicts the overall Sesame architecture. A central concept in the Sesame framework is the repository - a storage container for RDF. This can mean a set of Java objects in memory, or it can mean a relational database.

Sesame supports RDF Schema inferencing. Given a set of RDF and/or RDF Schema, Sesame can find the implicit information in the data. Sesame supports this by adding all implicit information to the repository as well when data is being added. Inferencing is supported only by a subset of repositories.

The Storage And Inference Layer, or SAIL API, is an internal API that abstracts from the storage format used, and provides reasoning support. SAIL implementations can also be stacked on top of each other, to provide functionality such as caching or concurrent access handling. Each Sesame repository has its own SAIL object to represent it. On top of the

SAIL, we find Sesame's functional modules, such as query engines, the admin module, and RDF export. Access to these functional modules is available through Sesame's Access APIs. Access APIs provide an access to Sesame functional modules. The Repository API provides high-level access to Sesame repositories, such as querying, storing of RDF files, extracting RDF, etc. The Graph API provides more fine-grained support for RDF manipulation, such as adding and removing individual statements, and creation of small RDF models directly from code.

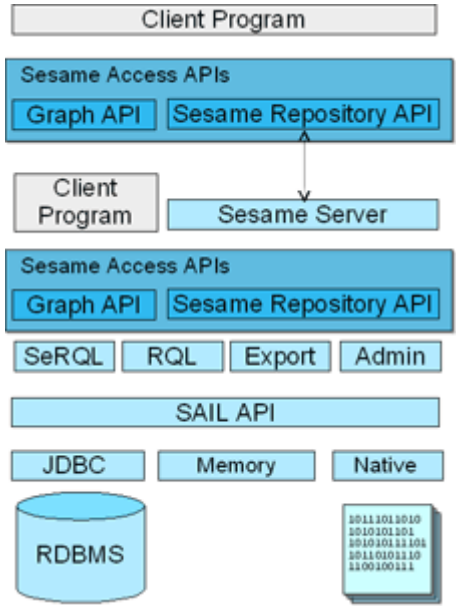

Fig. 1. Sesame Architecture

The Access APIs provide direct access to Sesame's functional modules, either to a client program, or to the next component of Sesame's architecture, the Sesame server. This is a component that provides HTTP-based access to Sesame's APIs. Then, on the remote HTTP client side, we again find the access APIs, which can again be used for communicating with Sesame as a server running on a remote location.

3.2 Jena

Jena is Semantic Web programmers' toolkit built upon RGF graph APIs. The Jena Framework includes: a RDF API, an OWL API, in-memory and persistent storage and SPARQL query engine.

The heart of the Jena2 architecture is the Graph layer containing the RDF graph. This layer is minimal by design, possible functionality is done in other layers. This permits a range of implementations of this layer such as in-memory or persistence triple stores.

The EnhGraph layer is the extension point on which to build APIs: within Jena2 the functionality offered by the EnhGraph layer is used to implement the Jena Model API and the new Ontology functionality for OWL and RDFS, upgrading the Jena DAML API.

I/O is done in the Model layer, essentially for historical reasons.
The Jena2 architecture supports fast path query that goes all the way through the layers from RDQL at the top right through to an SQL database at the bottom, allowing user queries to be optimized by the SQL query optimizer.
The Graph layer provides:

- triple stores, both in memory and backed by persistent storage;
- read-only views of non-triple data as triples, such as data read from a local file system, or scraped from a web page;
- virtual triples corresponding to the results of inference processes over some further set of triples.

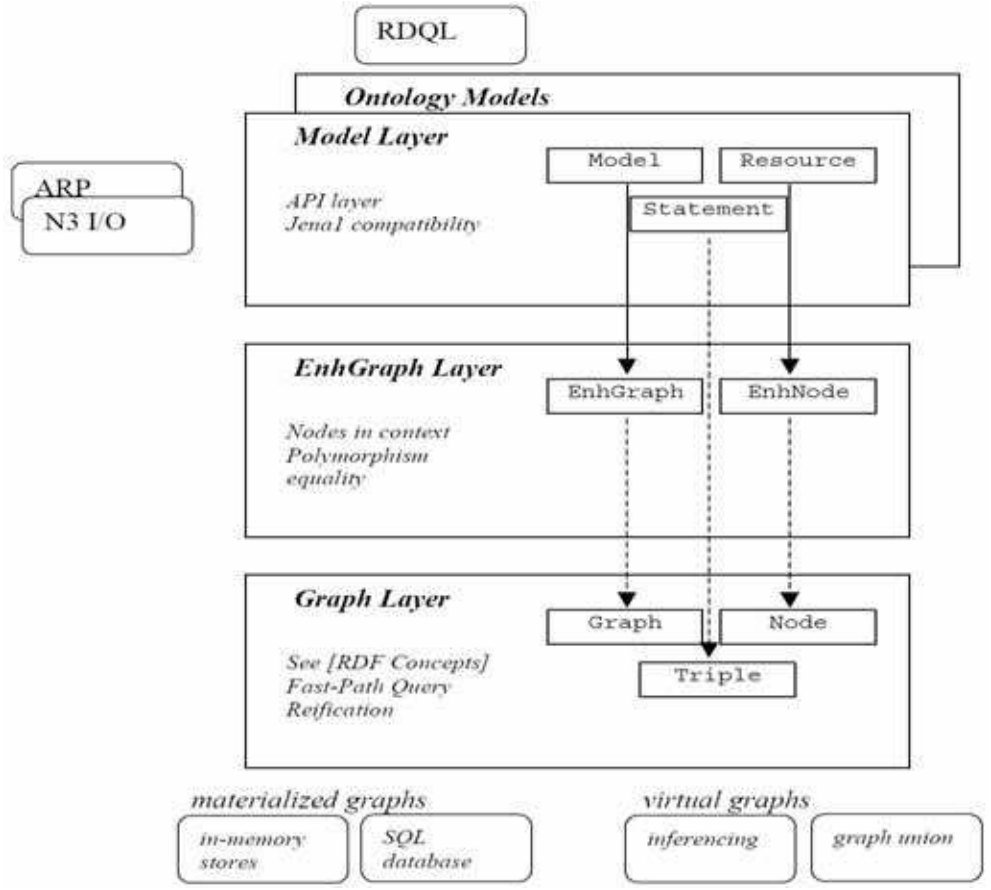

Fig. 2. Jena Architecture

Implementations of the Graph layer give a variety of concrete (materialized) triple stores, and built-in inference for RDFS and a subset of OWL.

The EnhGraph layer is designed to permit multiple views of graphs and nodes which can be used simultaneously. This allows the multiple inheritance and typing of RDFS to be reflected even in Java's single inheritance model.
The Model Layer maintains the Model API as the primary abstraction of the RDF graph used by the application programmer. This gives a much richer set of methods for operating on both the graph itself (the Model interface) and the nodes within the graph (the Resource interface and its subclasses). Further, the Ontology API can be realized as a DAML API or an OWL API.
The design goals for the Graph layer include:

- allowing collections of triples to be queried and updated efficiently. In particular, querying triples held in databases should be able to exploit the underlying database engine.
- being easy to reimplement, so that new triple collections can be represented by Graphs with minimal programming effort.
- supporting some specialist operations from the Model API when these cannot be easily constructed from the base functionality, reification in particular.

The elements within a Graph are Triples; each Triple comprises three Nodes, the subject, predicate, and object fields. A Node represents the RDF notion of a URI label, a blank node, or a literal; there are also two variable nodes, for named variables and a match-anything wildcard, for use in the Query interface.
The RDF restrictions that a literal can only appear as an object, and that a property can only be labelled with a URI, are not enforced by the Graph layer but by the Model layer. The core Graph interface supports modification (add and delete triples) and access (test if a triple is present or list all triples present matching some pattern). Graph implementations are free to restrict the particular triples they regard as legal and to restrict

3.3 Mulgara

Mulgara is a monolithic special-purpose database engine. The data access is provided using iTQL (Interactive Tucana Query Language).
Mulgara has an open API that supports various industry-standard programming languages and protocols. Different types of users interact with Mulgara in different ways depending on their needs:

- End users interact with Mulgara indirectly via applications that use Mulgara as the underlying data repository.
- System administrators use iTQL to load metadata into Mulgara, check its status, back up information held, or otherwise administer Mulgara databases.
- Programmers perform the integration between their own applications and Mulgara.

Queries to Mulgara databases can be issued alternatively using following mechanisms: iTQL shell, Simple Object Access Protocol (SOAP), iTQL JavaBean, or Mulgara Driver. Each of the above mechanisms connect to a Mulgara driver, which in turn connects to a Mulgara server over a separate communication channel. The communication channel is configurable and determines how a server exposes itself to the world, generally via RMI or SOAP.
Using SOAP allows Mulgara to run on a different server from an organizations' web application server, and still maintain accessibility through the regular corporate firewall. If this is not required (for example, the Java RMI port is opened on the firewall, or the Mulgara

server and the web application server are running on the same machine) the JSP tag libraries can communicate with the driver directly using the iTQL bean, effectively removing a level of indirection. The JSP tag libraries and COM object provide convenient methods by which iTQL commands can be sent and answers parsed into suitable formats.
iTQL commands received by the SOAP endpoint are sent to the iTQL interpreter. The interpreter uses a driver to issue commands to Mulgara servers containing the models specified by the command. The driver is responsible for coordinating the distribution of commands to the appropriate servers, and collating the results into a single answer to be returned to the interpreter. The remote server receives method calls from the driver via RMI, and makes the appropriate method calls to the underlying Mulgara database.
Pluggable resolvers, provided with Mulgara or written by third parties, allow Mulgara to query different data source, including:

- local or distributed Mulgara native XA datastore
- Lucene models
- XSD datatype models
- Views
- External data sources such as relational databases or Mbox files

3.4 Redland

Unlike almost all other relevant projects that are written in Java, Redland RDF libraries are implemented in the C language. Its APIs enable manipulation with the RDF graph, triples, URIs and Literals. The RDF graphs can be stored either in memory or persistently using Sleepycat/Berkeley DB, MySQL, PostgreSQL, AKT Triplestore, SQLite, files or URIs. It supports multiple syntaxes for reading and writing RDF as RDF/XML, N-Triples and Turtle Terse RDF Triple Language, RSS and Atom syntaxes, and for querying with SPARQL and RDQL using the Rasqal RDF Query Library.

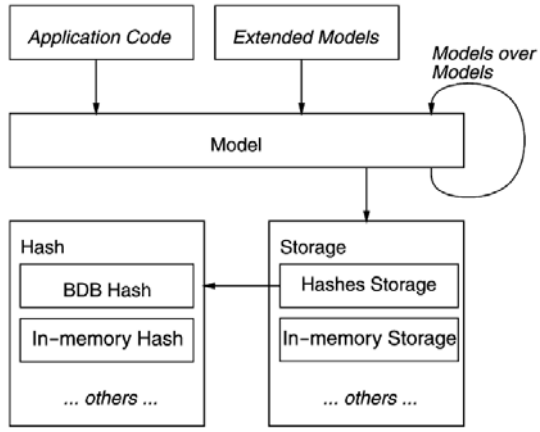

Fig. 3. Redland model layers

The library provides numerous bindings to other programming languages including Java, C#, Perl, PHP, Python and Ruby. The library is supplemented by a set of command line utilities for processing, parsing and querying RDF.

4. Query Languages and Reasoning

The ability to query the data stored in the repository is one of the key functions provided by all of the systems. However, the concrete implementation of this general task differs greatly from system to system. The systems usually provide some graph API to handle RDF graphs directly, but something more advanced is required for efficient application development. This role is performed by query languages, which define textual representation and more importantly the semantics of queries that the user can use to access data stored in the repository. A good parallel is the role of SQL in relational databases.
But since the very beginning, the Semantic web was supposed to do more than just return the data that was loaded into the database earlier. Semantic repositories are supposed to provide reasoning, i.e. the ability to infer new statements (knowledge) based on the statements that were loaded into the database. To do so, several technologies were developed or adopted for the use in the Semantic web. The most prominent are RDFS and OWL, the former being able to construct type hierarchies for resources and statements while the latter is a much more complex system based on the idea of description logics. The intricacy of OWL together with high time complexity results in varying degrees of adoption in different systems.

4.1 Sesame

Sesame supports two distinct query languages: SeRQL and SPARQL. SeRQL is a language developed by the Sesame team before the SPARQL standard was created. Its syntax is SQL-like, even more so than in the case of SPARQL. Even some advanced constructs of SQL have their parallels in SeRQL, for instance set operators (UNION, MINUS, INTERSECT, IN or EXISTS). The following query is an example of SeRQL:

```
SELECT DISTINCT
   label(ArtefactTitle), MuseumName
FROM
   {Artefact} arts:created_by {} arts:first_name {"Rembrandt"},
   {Artefact} arts:exhibited {} dc:title {MuseumName},
   {Artefact} dc:title {ArtefactTitle}
WHERE
   isLiteral(ArtefactTitle) AND
   lang(ArtefactTitle) = "en" AND
   label(ArtefactTitle) LIKE "*night*"
USING NAMESPACE
   dc   = <http://purl.org/dc/elements/1.0/>,
   arts = <http://example.org/arts/>
```

Fig. 4. SeRQL example

Overall, it is a pragmatic query language with clear syntax and relatively simple definition. Its complete specification is a part of the Sesame documentation. SeRQL is a viable option in cases where SPARQL compatibility is not necessary.
But Sesame also provides very advanced SPARQL implementation. According to W3C report (Harris 2008), it covers the whole specification.
Sesame only (optionally) supports RDFS reasoning. It is important to note that it is not handled by the "core" Sesame. Inferencing (reasoning) is supposed to be handled by the underlying storage system. This means that the RDFS reasoning is actually feature of the in-memory and native RDF repositories provided by Sesame and that different modules can provide more complex reasoning. One such example is OWLIM (http://www.ontotext.com/owlim), which implements the SAIL API and provides OWL reasoning.

4.2 Jena

Jena – the ARQ module, to be exact – provides full support for SPARQL (complete coverage according to W3C). It also offers several significant extensions that go beyond the SPARQL specification. These include:

- property paths – regular expressions over paths in the RDF graph,
- querying remote SPARQL endpoints – a query can specify URI of an endpoint that should be queried,
- group by, having and aggregation – very similar to SQL,
- selection list with expressions – queries can return computed values, e.g. sum of two variables,
- nested SELECTs – allows the use of sub-queries, especially powerful in combination with two extensions mention just above, and
- variable assignment – values of variables can be computed by an expression from other variable values.

Jena also implements the SPARQL/Update proposal, which allows insertion and deletion of values stored in the RDF store.
Jena supports RDFS and OWL reasoning "out of the box". Like in Sesame, other reasoners can be added thanks to an API provided by Jena. But unlike Sesame, Jena inference API does not combine storage with reasoning. It can even support stacked reasoners.
Built in reasoners are:

- transitive rasoner, which handles only subClassOf and subPropertyOf,
- RDFS reasoner, and
- Rubrik reasoner, which supports rule-based inferencing.

Jena contains RDFS and OWL Lite rule sets for the Rubrik reasoner, but application specific sets can be used as well.
Examples of external reasoners include the popular Racer reasoner or FaCT.

4.3 Mulgara

Mulgara supports two different query languages: SPARQL and TQL. The TQL language is specific to the Mulgara system and its syntax is also SQL-like. It offers some advanced features like graph traversing, transitive closure, set operations, ordering or sub-queries. It

also provides commands that allow the user to update the data and even to configure and manage the database. Unfortunately, the provided documentation is rather brief.
The SPARQL implementation is not complete (although thorough implementation report is unavailable) but the development team is improving it continually.
Mulgara uses the Krule rule engine developed as part of the original Kowari system. It is a general entailment system built around RLog logic language. The rule sets are then applied to graphs (data) in the system. An RDFS rule set is available in the Mulgara sources.

4.4 Redland

Redland provides the Rasqal RDF query library with support for SPARQL and RDQL. It is only partial implementation of the SPARQL standard but full support of RDQL. While the development of Rasqal seems to have slowed down over the last years, the project is still active and new releases are still being published.
Since Redland is designed purely as an RDF store, no reasoning is supported.

5. Data Engine

While the provided query capabilities are an important factor and define the system's ability to extract data, the storage of the data is also an important factor. It greatly affects the performance of queries and in some cases even influences the query capabilities by providing inferencing mechanism as part of the storage. It also defines system's ability to handle transactions and failure recovery.

5.1 Sesame

Sesame comes packed with two kinds of specialized native storage for RDF, two kinds of RDF storage built on top of a relational database and a general connection to a remote RDF repository.
Both native RDF storages offer different levels of inference. The first native RDF repository is an in-memory store. It offers very fast access to data, but has two serious disadvantages: it is not persistent and it takes a lot of memory. The second disadvantage becomes clearly visible when working with data of an average size (90MB RDF N3 file), where the memory has been exhausted and the whole Sesame system crashed including Apache Tomcat server.
The second native RDF storage is simply called "Native Java Store". It is persistent, offers very good performance, and consumes reasonable amount of memory. It is able to load and query larger data.
The second possibility is to use a relational RDF storage. Out of the box, Sesame supports two connectors to relational database engines common on Linux platforms: MySQL and PostgreSQL. Both of them offer reasonable performance, they are able to store large data, and they have decent memory requirements.
The last possibility is a connection to any RDF repository satisfying a HTTP-based protocol for Sesame, which is partially described in Sesame system documentation.
Other repositories can be added using the SAIL API. At least one notable example should be mentioned: OWLIM Semantic repository, which offers high-performance semantic repository services. A commercial version called BigOWLIM is available as well, which claims to operate upon billions of triples.

5.2 Jena

Jena itself is only a framework and the user needs to install additional storage subsystem. As usually, there are two kinds of storages: native RDF storage and SQL-based storages. Both subsystems are directly referenced from the Jena web site.

The native Java storage for Jena is called TDB and it performs very well. It is able to load and query large datasets and authors claim that it is able to operate upon billions of triples. Interestingly, the TDB behaves differently on 32-bit and 64-bit JVMs, where 32-bit version handles caching by itself, while 64-bit version utilizes memory-mapped files and caching is left to the operating system.

The second possibility is SDB storage, which is a storage subsystem built on top of a SQL database server. SDB supports variety of underlying SQL servers including commercial ones like Microsoft SQL Server or Oracle. Connection to the SQL server from SDB is described in one simple configuration file. Underlying SQL server brings robustness and well known practices for managing, maintenance, and backup of SQL databases, but at the price of lower performance in data loading, where known limitations of SQL servers arise.

5.3 Mulgara

Mulgara is primarily a native RDF database in Java, so the development is focused on storage and querying. As a successor of Kowari, it is high-performance RDF storage. Unfortunately, there is no documentation to the Mulgara internals. As the FAQ for Mulgara states, there are three possibilities, all of them quite inadvisable: read the source code, ask developers, or read Paul's blog.

5.4 Redland

Redland offers two kinds of storage engines: in-memory temporal storage with very fast query and access times and persistent storage with quite decent list of SQL database backends. The advantages and disadvantages of SQL backends are the same as in Jena.

5.5 Virtuoso

So far, we only dealt with the four systems and their different components. But since most of them are built as modular systems, some parts can be switched for modules, even from different authors. This is mostly used for data engines and OpenLink Virtuoso is a nice example of such module.

Virtuoso is a cross platform server to implement web, file, and database server functionality alongside native XML storage, and universal data access middleware. It supports many contemporary internet technologies and protocols as well as operating systems. Virtuoso provides transparent access to existing data sources, which are typically databases from different database vendors. Figure 5 depicts how applications that are built in conformance with industry standards. Virtuoso exposes its functionality to Web Services, it provides specialized providers for Sesame, Jena and Mulgara frameworks.

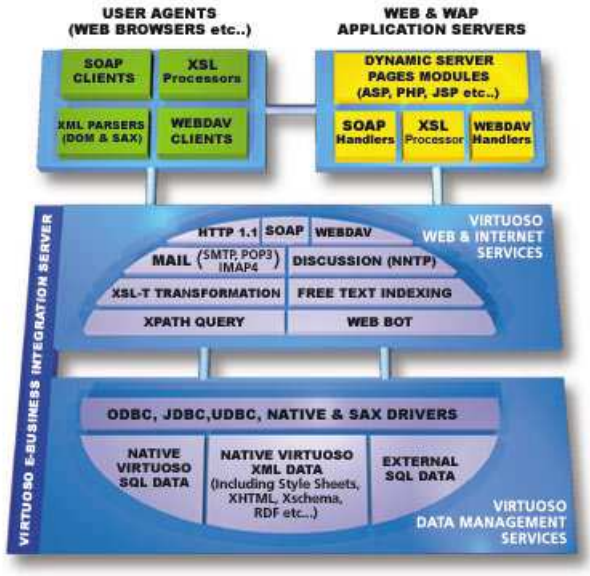

Fig. 5. Virtuoso architecture

6. Installation and Documentation

Most resources (papers and web pages) deal with performance, query languages, APIs and interfaces, but there are two other important aspects of any software. In order to operate such systems, one first has to install it. Since most Semantic web projects are built on top of many different libraries, the ease of installation may become a significant concern. The other aspects that often tends to be neglected by such projects is maintaining detailed and up-to-date documentation.

6.1 Sesame

The Sesame is delivered as a Java archive in two flavours. One of them is called "onejar" and it is a distribution best suitable for already compiled applications written for Sesame. The second flavour of the distribution is SDK, which contains all sources needed for developing Sesame applications as well as two .war files, which can be unpacked into Apache Tomcat application directory, where they create two applications: Sesame server with HTTP interface and Workbench, which is a web interface for playing with Sesame. We have to discover some nontrivial installation steps, like copying latest Apache Xalan-J distributable files into library directory of Tomcat, because the installation documentation is clearly outdated and short, and there is no notion about Xalan-J installation.
The documentation is in the form of HTML pages, either distributed in SDK distribution or reachable on the web pages of the Sesame project.

6.2 Jena

As mentioned above, the Jena is only a framework and for correct implementation more packages must be installed, although some of them are already contained in Jena downloadable package. All downloads are easily accessible from the Jena download page. At least two components are required for correct installation, the Jena framework and a storage subsystem. Currently there are two options: TDB and SDB. As an optional but highly recommended component is the Joseki package, which serves as an RDF publishing server.

The installation is smooth and easy as it only requires the user to set some environmental variables including CLASSPATH and sometimes edit a small number of configuration files. Moreover, the installation and configuration for each component is described in the project documentation very well.

The Jena documentation consists of web pages of the project, where each subproject/component has its own documentation.

6.3 Mulgara

Installation should be quite easy and straightforward, as it is shipped with variety of packages for different usage scenarios. The user can choose from complete package, package without third party dependencies, Web Archives, etc.

We have encountered interesting problem, as the current (at the time of writing of this text) version 2.0.9 had broken Web Archive distribution, but following versions 2.1.0 and 2.1.1 repaired this problem.

The documentation is rather brief, but it describes everything required to install and run the system.

6.4 Redland

Redland library requires two other libraries, Raptor and Rasqal, for compilation and runtime as well, and we will call these three libraries Redland libraries from now. All libraries are available for download at the Redland project page. The Redland libraries are (unlike all previously mentioned projects) written in C and are distributed in source code form with automake build system. Although creating make configuration using automake should be simple from user's point of view, in this case we have encountered some nontrivial problems.

The first problem has been caused by requirements on newer versions of some tools like flex, libtool, etc. We have used the latest RedHat Enterprise Linux 5.3 (RHEL) distribution and the required versions of the tools are not available. Manual installation of newer version has caused some dependency problems with existing software packages on RHEL.

The second problem has been linked to pkg-config tool, which couldn’t find the already compiled Raptor and Rasqal libraries and failed to create configuration for Redland. Manual steps were required to correct this situation.

This somewhat limits the audience the library can target. It clearly isn’t prepared for enterprise deployment, where stable versions of Linux (like RHEL) or Unix are usually used.

The documentation is surprisingly comprehensive and is provided separately for each library.

7. Other Projects

There are many other RDF repositories. We have only listed some of the most important ones. The others are either not as popular or their development was discontinued. One such example is the Kowari system (Wood 2005). While it was one of the earliest and very efficient RDF store with good scalability, its development stopped in 2005. Fortunately, the code base was later used to create the Mulgara project, which is still being very active.
Many of the projects have been created only as a scientific prototype, which means their development may easily stop at any time and their support can be very problematic. Examples include TAP from Stanford (Guha and McCool, 2003), Corese from INRIA (Corby et al., 2004), Trisolda (Dokulil et al., 2009), or Boca (Feigenbaum et al., 2007).
Comprehensive list of Semantic Web tools, including RDF repositories, is maintained by W3C (Alexander 2009). Unfortunately, it does not give current state of development of the individual systems and whether they are still being developed and supported.

8. Conclusion

Over time, a lot of projects aimed to be the framework for the Semantic Web. Only a few became stable and ready to use for diverse semantization projects. In this survey we described four such systems, their architecture, capabilities, strengths and limitations. Based on our practical experience, although these projects are not meant to became enterprise-level frameworks (e.g. none of them support access rights), they can be used as a platform for research of semantization.

9. References

Alexander, K. (2009) Semantic Web Development Tools, *http://esw.w3.org/topic/SemanticWebTools*, W3C Technical Report, 2009

Beckett, D. (2002). The design and implementation of the Redland RDF application framework, *Computer Networks*, Vol. 39, Issue 5, August 2002, pp. 577-588

Broekstra, J.; Kampman, A.; van Harmelen, F. (2002) Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema, *The Semantic Web — ISWC 2002*, Lecture Notes in Computer Science, vol. 2342, 2002, ISBN 978-3-540-43760-4

Carrol, J. J.; Reynolds, D.; Dickinson, I.; Seaborne, A.; Dollin, C.; Wilkinson, K. (2004). Jena: Implementing the Semantic Web Recommendations, *Proceedings of the 13th international World Wide Web conference*, 2004, New York, ISBN:1-58113-912-8

Corby, O.; Dieng-Kuntz, R. and Faron-Zucker, C. (2004) Querying the Semantic Web with Corese Search Engine, *Proc. 16th European Conf. Artificial Intelligence* (ECAI 04), 2004, IOS Press, pp. 705–709.

Dokulil, J.; Yaghob, J. and Zavoral, F. (2009) Trisolda: The Environment for Semantic Data Processing, *International Journal On Advances in Software*, 2008, vol. 1, IARIA, 2009

Feigenbaum, L.; Martin, S.; Roy, M. N.; Szekely, B. and Yung, W. C. (2007) Boca: an open-source RDF store for building Semantic Web applications, Briefings in Bioinformatics, Vol. 8, Issue 3, pp. 195-200, ISSN 1467-5463

Guha, R.; McCool, R. (2003) TAP: a Semantic Web platform, *Computer Networks*, Vol. 42, Issue 5, 2003, pp. 557-577

Harris, S. (2008) SPARQL Implementation Coverage Report, W3C Technical Report, *http://www.w3.org/2001/sw/DataAccess/tests/implementations*, 2008

Muys, A. (2006) Building an Enterprise-Scale Database for RDF Data, *Netymon technical paper*, 2006

Prud'hommeaux, E.; Seaborne, A. (2008) SPARQL Query Language for RDF, W3C Recommendation, W3C 2008

Wood D. (2005) Scaling the Kowari Metastore, *Lecture Notes in Computer Science,* Vol. 3807, pp. 193-198, Springer Verlag, 2005

Semantic Web
Edited by Gang Wu

ISBN 978-953-7619-54-1
Hard cover, 310 pages
Publisher InTech
Published online 01, January, 2010
Published in print edition January, 2010

Having lived with the World Wide Web for twenty years, surfing the Web becomes a way of our life that cannot be separated. From latest news, photo sharing, social activities, to research collaborations and even commercial activities and government affairs, almost all kinds of information are available and processible via the Web. While people are appreciating the great invention, the father of the Web, Sir Tim Berners-Lee, has started the plan for the next generation of the Web, the Semantic Web. Unlike the Web that was originally designed for reading, the Semantic Web aims at a more intelligent Web severing machines as well as people. The idea behind it is simple: machines can automatically process or “understand” the information, if explicit meanings are given to it. In this way, it facilitates sharing and reuse of data across applications, enterprises, and communities. According to the organisation of the book, the intended readers may come from two groups, i.e. those whose interests include Semantic Web and want to catch on the state-of-the-art research progress in this field; and those who urgently need or just intend to seek help from the Semantic Web. In this sense, readers are not limited to the computer science. Everyone is welcome to find their possible intersection of the Semantic Web.

How to reference
In order to correctly reference this scholarly work, feel free to copy and paste the following:

Jiri Dokulil, Jakub Yaghob and Filip Zavoral (2010). Semantic Infrastructures, Semantic Web, Gang Wu (Ed.), ISBN: 978-953-7619-54-1, InTech, Available from: http://www.intechopen.com/books/semantic-web/semantic-infrastructures

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821