Sesame: An Architecture for Storing and Querying RDF Data and Schema Information

Jeen Broekstra
Aidministrator Nederland b.v.
jeen.broekstra@aidministrator.nl

Arjohn Kampman
Aidministrator Nederland b.v.
arjohn.kampman@aidministrator.nl

Frank van Harmelen
Faculty of Sciences, Vrije Universiteit Amsterdam
frank.van.harmelen@cs.vu.nl

Abstract

RDF and RDF Schema provide the first W3C standard to enrich the Web with machine-processable semantic data. However, to be able to use this semantic data, a scalable, persistent RDF store and a powerful query engine using an expressive query language are needed.

Sesame is an extensible architecture implementing both of these. Sesame can be based on arbitrary repositories, ranging from traditional Data Base Management Systems, to dedicated RDF triple stores.

Sesame also implements a query engine for RQL, the most powerful RDF/RDF Schema query language to date.

1 Introduction

The Resource Description Framework (RDF) [Lassila and Swick, 1999] is a W3C Recommendation for the notation of meta-data on the World Wide Web. RDF Schema [Brickley and Guha, 2000] extends this standard by providing developers with the means to specify vocabulary and to model object structures. These techniques will enable the enrichment of the Web with machine-processable semantics, thus giving rise to what has been dubbed the Semantic Web.

However, simply having this data available is not enough. Tooling is needed to process the information, to transform it, to reason with it. As a basis for this, we have developed Sesame, an architecture for efficient storage and expressive querying of large quantities of RDF meta-data. Sesame is being developed by Aidministrator Nederland b.v.[1] as part of the European IST project On-To-Knowledge[2] [Fensel et al., 2000].

This paper is organized as follows: in section 2 we give a short introduction to RDF and RDF Schema.

This section is only to make the paper self-contained, and can be skipped by readers already familiar with these languages.

In section 3 we discuss why a query language specifically tailored to RDF and RDF Schema is needed,

over and above existing query languages such as XQuery. In section 4 and 5 we look in detail at Sesame's architecture. Section 6 discusses our experiences with Sesame until now, and section 7 looks into possible future developments. Finally we provide our conclusions in section 8.

2 RDF and RDF Schema

The Resource Description Framework (RDF) [Lassila and Swick, 1999] is a W3C recommendation that was originally designed to standardize the definition and use of metadata-descriptions of Web-based resources. However, RDF is equally well suited to representing arbitrary data, be they meta-data or not.

[1] See `http://www.aidministrator.nl/`
[2] See `http://wwww.ontoknowledge.org/`

2.1 RDF

The basic building block in RDF is an object-attribute-value triple, commonly written as $A(O, V)$. That is, an object O has an attribute A with value V. Another way to think of this relationship is as a labeled edge between two nodes: $[O] - A \rightarrow [V]$.

This notation is useful because RDF allows objects and values to be interchanged. Thus, any object from one triple can play the role of a value in another triple, which amounts to chaining two labeled edges in a graphic representation. The graph in figure 1 for example, expresses the following relationships:

```
hasName
    ('http://www.famouswriters.org/twain/mark',
     "Mark Twain")
hasWritten
    ('http://www.famouswriters.org/twain/mark',
     'http://www.books.org/ISBN0001047582')
title
    ('http://www.books.org/ISBN0001047582',
     "The Adventures of Tom Sawyer")
```

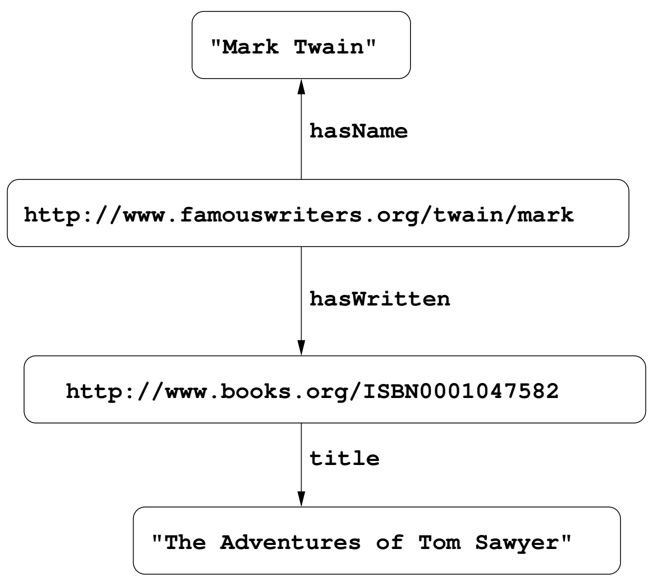

Figure 1: An example RDF data graph, capturing three statements

RDF also allows a form of reification[3] in which any RDF statement itself can be the object or value of a triple. This means graphs can be nested as well as chained. On the Web this allows us, for example, to express doubt or support for statements created by other people. Finally, it is possible to indicate that a given object is of a certain type, such as stating that "ISBN0001047582" is of the type Book, by creating a type edge referring to the Book definition in an RDF schema:

```
type
    ('http://www.books.org/ISBN0001047582',
    'http://www.description.org/schema#Book')
```

The RDF Model and Syntax specification also proposes an XML syntax for RDF data models. One possible serialisation of the above relations in this syntax, would look like this:

[3] *rei facere (Lat.)* 'to make into a thing'

```
<rdf:Description rdf:about="http://www.famouswriters.org/twain/mark">
    <s:hasName>Mark Twain</s:hasName>
    <s:hasWritten rdf:resource="http://www.books.org/ISBN0001047582"/>
</rdf:Description>

<rdf:Description rdf:about="http://www.books.org/ISBN0001047582">
    <s:title>The Adventures of Tom Sawyer</s:title>
    <rdf:type rdf:resource="http://www.description.org/schema#Book"/>
</rdf:Description>
```

Since the proposed XML syntax allows many alternative ways of writing down information (and indeed still other syntaxes may be introduced), the above XML syntax is just one of many possibilities of writing down an RDF model in XML.

It is important to note that RDF is designed to provide a basic object-attribute-value model for Web-data. Other than this intentional semantics – described only informally in the standard – RDF makes no data modeling commitments. In particular, no reserved terms are defined for further data modeling. As with XML, the RDF data model provides no mechanisms for declaring property names that are to be used.

2.2 RDF Schema

RDF Schema [Brickley and Guha, 2000] is a mechanism that lets developers define a particular vocabulary for RDF data (such as hasWritten) and specify the kinds of objects to which these attributes can be applied (such as Writer). RDF Schema does this by pre-specifying some terminology, such as Class, subClassOf and Property, which can then be used in application-specific schemata. RDF Schema expressions are also valid RDF expressions – in fact, the only difference with 'normal' RDF expressions is that in RDF Schema an agreement is made on the *semantics* of certain terms and thus on the *interpretation* of certain statements. For example, the subClassOf property allows the developer to specify the hierarchical organization of classes. Objects can be declared to be instances of these classes using the type property. Constraints on the use of properties can be specified using domain and range constructs.

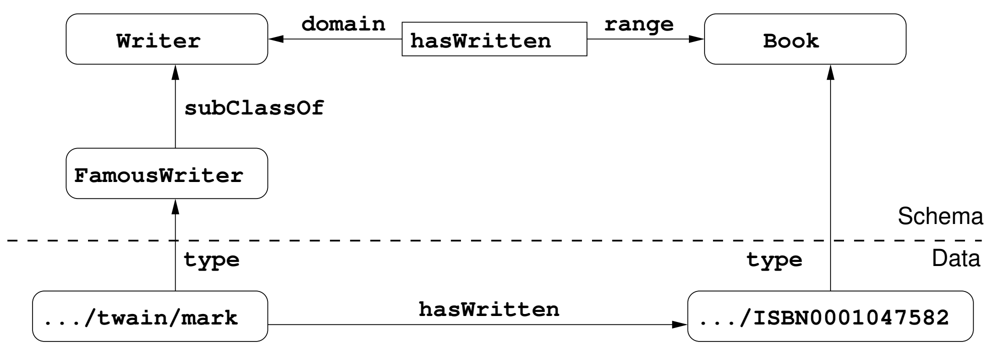

Figure 2: An example RDF Schema, defining vocabulary and a class hierarchy

Above the dotted line in figure 2, we see an example RDF schema that defines vocabulary for the RDF example we saw earlier: Book, Writer and FamousWriter are introduced as classes, and hasWritten is introduced as a property. A specific instance is described in terms of this vocabulary below the dotted line.

3 The need for an RDF/S Query Language

RDF documents and RDF schemata can be considered at three different levels of abstraction:

1. at the *syntactic level* they are XML documents
2. at the *structure level* they consist of a set of triples
3. at the *semantic level* they constitute one or more graphs with partially predefined semantics.

We can query these documents at each of these three levels. We will briefly consider the pros and cons of doing so for each level in the next sections. This will lead us to conclude that RDF(S) documents should really be queried at the semantic level. We will briefly discuss RQL, a language for querying RDF(S) documents at the semantic level.

3.1 Querying at the syntactic level

As we have seen in section 2, any RDF model (and therefore any RDF schema) can be written down in XML notation. It would therefore seem reasonable to assume that we can query RDF using an XML query language (for example, XQuery [Chamberlin et al., 2001]).

However, this approach disregards the fact that RDF is not just an XML dialect, but has its own data model that is very different from the XML tree structure. Relationships in the RDF data model that are not apparent from the XML tree structure become very hard to query.

As an example, let us look again at the XML description of the RDF model in figure 1.

```
<rdf:Description rdf:about="http://www.famouswriters.org/twain/mark">
    <s:hasName>Mark Twain</s:hasName>
    <s:hasWritten rdf:resource="http://www.books.org/ISBN0001047582"/>
</rdf:Description>

<rdf:Description rdf:about="http://www.books.org/ISBN0001047582">
    <s:title>The Adventures of Tom Sawyer</s:title>
    <rdf:type rdf:resource="http://www.description.org/schema#Book"/>
</rdf:Description>
```

In an XML query language such as XQuery [Chamberlin et al., 2001], expressions to traverse the data structure are tailored towards traversing a node-labeled tree. However, the RDF data model in this instance is a graph, not a tree, and moreover, both its edges (properties) and its nodes (subjects/objects) are labeled. In querying at the syntax level, this is literally left as an excercise for the query builder: one cannot query the relation between the resource signifying ‘Mark Twain’ and the resource signifying ‘The Adventures of Tom Sawyer’ without knowledge of the syntax that was used to encode the RDF data in XML.

Ideally, we would want to formulate a query like “Give me all the relationships that exist between Mark Twain and The Adventures of Tom Sawyer”. However, using only the XML syntax, we are stuck with formulating an awkward query like “Give me all the elements nested in a Description element with an about attribute with value ’http://www.famouswriters.org/twain/mark’, of which the value of its resource attribute occurs elsewhere as the about attribute value of a Description element which has a nested element title with the value ‘The Adventures of Tom Sawyer’.”

Not only is this approach inconvenient, it also disregards the fact that the XML syntax for RDF is not unique: different ways of encoding the same information in XML are possible and in use currently. This means that one query will never be guaranteed to retrieve all the answers from an RDF model.

3.2 Querying at the structure level

When we abstract from the XML linearisation syntax, any RDF document represents a set of triples, each triple representing a statement of the form Object-Attribute-Value. A number of query languages have been proposed and implemented that regard RDF documents as such a set of triples, and that allow to query such a triple set in various ways. See `http://perso.enst.fr/˜ta/web/rdf/rdf-query.html` for a recent overview.

The RDF/RDF Schema example from figure 2 corresponds to the following set of triples:

```
(type Book Class)
(type Writer Class)
(type FamousWriter Class)
(subClassOf FamousWriter Writer)
(type hasWritten Property)
(domain hasWritten Writer)
(range hasWritten Writer)
(type twain/mark FamousWriter)
(type ISBN0001047582 Book)
(hasWritten twain/mark ISBN0001047582)
```

An RDF query language would allow us to query which resources are known to be of type FamousWriter:

```
select ?x from ... where (type ?x FamousWriter)
```

The clear advantage of such a query is that it directly addresses the RDF data model, and that it is therefore independent of the specific XML syntax that has been chosen to represent the data.

However, a major shortcoming of any query-language at this level is that it interprets *any* RDF only as a set of triples, including those elements which have been given a special semantics in RDF Schema. For example, since http://www.famouswriters.org/twain/mark is of type FamousWriter, and since FamousWriter is a subclass of Writer, http://www.famouswriters.org/twain/mark is also of type Writer, by virtue of the intended RDF Schema semantics of type and subClassOf. However, there is no triple that explicitly asserts this fact. As a result, the query

```
SELECT ?x
FROM   (type ?x Writer)
```

will fail because the query only looks for explicit triples in the store, whereas the triple `(type /twain/mark Writer)` is not explicitly present in the store, but is implied by the semantics of RDF Schema. Notice that simply expanding the query into something like

```
SELECT ?x ?c1 ?c2 ?c3
FROM   (type ?x ?c1),
       (subClassOf ?c2 ?c3)
WHERE  ?c1 = ?c2
```

will solve the problem in this specific example, but does not cater for a chain of subClassOf triples, etc.

3.3 Querying at the semantic level: RQL

What is clearly required is a query language that is sensitive to the semantics of the RDF Schema primitives. RQL [Karvounarakis et al., 2000, Alexaki et al., 2000] is the first (and to the best of our knowledge currently the only) proposal for a declarative query language for RDF and RDF Schema. It is being developed within the European IST project C-Web and its followup project MESMUSES by the Institute of Computer Science at FORTH, in Greece[4].

RQL adopts the syntax of OQL [Cattel et al., 2000]. As OQL, RQL is a functional language: the output of RDF Schema queries is again legal RDF Schema code, which allows the output of queries to function as input for subsequent queries.

RQL is defined by means of a set of core queries, a set of basic filters, and a way to build new queries through functional composition and iterators.

The core queries are the basic building blocks of RQL, which give access to the RDF Schema specific contents of an RDF triple store, with queries such as Class (retrieving all classes), Property (retrieving all properties) or Writer (returning all instances of the class with name Writer). This last query returns of course also all instances of subclasses of Writer, since these are also instances of the class Writer, by virtue of the semantics of RDF Schema. We can ask for all *direct* instances of Writer (i.e. ignoring all instances of subclasses) through the query ˆWriter.

RQL can also query the structure of the subclass hierarchy. In our example, the query subClassOf(Writer) would return the class FamousWriter as its only result. In general, this would return all direct and indirect subclasses of Writer, since RQL is aware of the transitivity of the subclass relation. The query subClassOfˆ(Writer) would return only the immediate subclasses.

Of course, being based on OQL, RQL also allows a select-from-where construct.

A final crucial feature of RQL are the path-expressions. These allow us to match patterns along entire paths in RDF/RDF Schema graphs, such as the one depicted in figure 2. For example, the query

```
SELECT Y
FROM FamousWriter {X}. hasWritten {Y}
```

returns all books written by famous writers, effectively doing pattern-matching along a path in the graph of figure 2.

[4] See `http://www.ics.forth.gr/`

3.4 Conclusion

The previous subsections have argued that RDF data should not be queried at the level of their (rather incidental) XML encoding, and that RDF Schema data should not be regarded as simply a set of RDF triples, since all intended semantics of the RDF Schema primitives are then lost. Consequently, we should be using a query language that is sensitive to this RDF Schema semantics. RQL is a powerful (and currently the only) candidate for such a language.

In the next few sections, we will discuss the architecture we have designed for a query engine for RQL.

4 Sesame's Architecture

The Sesame system is a Web-based architecture that allows persistent storage of RDF data and schema information and subsequent online querying of that information. In section 4.1, we present an overview of Sesame's architecture. In the sections following that, we look in more detail at several components.

4.1 Overview

An overview of Sesame's architecture is shown in Figure 3. In this section we will give a brief overview of the main components.

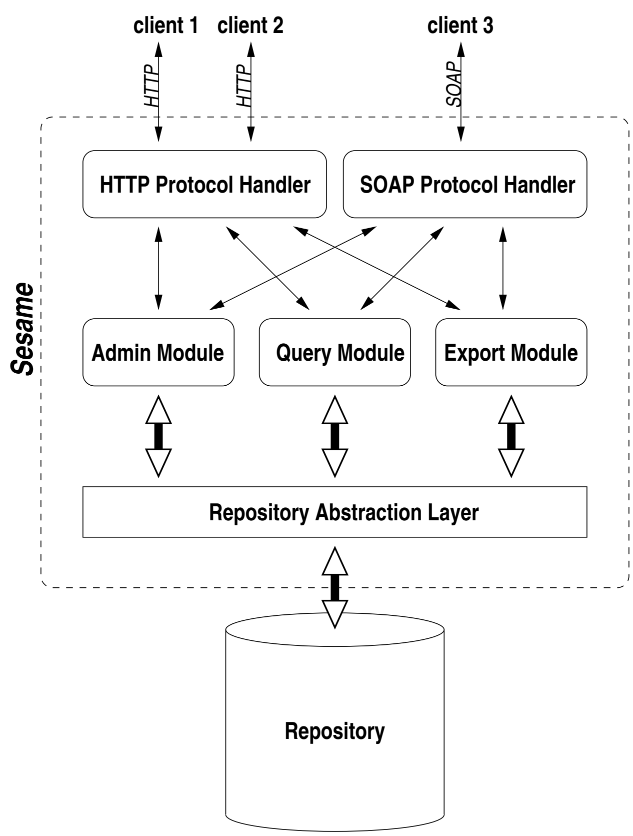

Figure 3: Sesame's architecture

For persistent storage of RDF data, Sesame needs a scalable repository. Naturally, a Data Base Management System (DBMS) comes to mind, as these have been used for decades for storing large quantities of data. In these decades, a large number of DBMS's have been developed, each having their own strengths and weaknesses, targeted platforms, and API's. Also, for each of these DBMS's, the RDF data can be stored in numerous ways.

As we would like to keep Sesame DBMS-independent and it is impossible to know which way of storing the data is best fitted for which DBMS, all DBMS-specific code is concentrated in a single architectural layer of Sesame: the *Repository Abstraction Layer* (RAL).

This RAL offers RDF-specific methods to its clients and translates these methods to calls to its specific DBMS. An important advantage of the introduction of such separate layer is that it makes it possible to implement Sesame on top of a wide variety of repositories without changing any of Sesame's other components. Section 4.3 describes a number of possible repository implementations.

Sesame's functional modules are clients of the RAL. Currently, there are three such modules:

- The RQL query module.
  This module evaluates RQL queries posed by the user (see section 5.1).

- The RDF administration module.
  This module allows incremental uploading of RDF data and schema information, as well as the deleting of information (see section 5.2).

- The RDF export module.
  This module allows the extraction of the complete schema and/or data from a model in RDF format (see section 5.3).

Depending on the environment in which it is deployed, different ways to communicate with the Sesame modules may be desirable. For example, communication over HTTP may be preferable in a Web context, but in other contexts protocols such as RMI (Remote Method Invocation)[5] or SOAP (Simple Object Access Protocol) [Box et al., 2000] may be more suited.

In order to allow maximal flexibility, the actual handling of these protocols has been placed outside the scope of the functional modules. Instead, protocol handlers are provided as intermediaries between the modules and their clients, each handling a specific protocol.

The introduction of the repository abstraction layer and the protocol handlers makes Sesame into a generic architecture for RDF (S) storage and querying, rather than just a particular implementation of such a system. Adding additional protocol handlers makes it easy to connect Sesame to different operating environments. The construction of concrete RAL's will be discussed in the next section.

Sesame's architecture has been designed with extensibility and adaptability in mind. The possibility to use other kinds of repositories has been mentioned before. Adding additional modules or protocol handlers is also possible. The only part that is fixed in the architecture is the RAL.

4.2 The Repository Abstraction Layer

As we have seen in the previous section, the Repository Abstraction Layer (RAL) offers a stable, high level interface for talking to repositories. This RAL is defined by an API that offers functionality to add data to, or to retrieve or delete data from the repository. RAL-implementations translate calls to the API methods into operations on the underlying repository.

Rather than adopting or extending an existing RDF API, such as the "Stanford API" proposed by Sergey Melnik [Melnik, 2000], we have created a completely new API.

The main differences between our proposal and the Stanford API are that:

a. The Stanford API is very much targeted at data that is kept in memory, whereas our API is considerably more "lightweight" as all data is returned one-at-a-time in data streams.

b. Our API supports RDF Schema semantics, such as subsumption reasoning, whereas the Stanford API only offers RDF-related functionality.

[5] See `http://java.sun.com/j2se/1.3/docs/guide/rmi/spec/rmiTOC.html`

The advantage of returning data in streams (point a) is that at any one time only a small portion of the data is kept in memory. This streaming approach is also used in the functional modules, and even in the protocol handlers which give results as soon as they are available. This approach is needed for Sesame to be able to scale to large volumes of data without requiring exceptionally expensive hardware. In fact, Sesame requires close to zero memory for data and only a small amount of memory for the program to run.

This, together with the option of using a remote data store for the repository (see section 4.3) makes Sesame potentially suitable for use as infrastructure in highly constrained environments such as portable devices.

Of course, reading everything from a repository and keeping nothing in memory seriously hurts performance. This performance problem can be solved by selectively caching data in memory[6]. For small data volumes it is even possible to cache all data in memory, in which case the repository only serves as a persistent storage. Sesame's architecture allows all of this to be done in a completely transparent way, as will be shown in the next section.

4.2.1 Stacking Abstraction Layers

An important feature of the RAL is that it is possible to put one on top of the other. To Sesame's functional modules (the admin, query and export modules) this is completely transparent, as they will only see the RAL at the top of the stack (see figure 4). The RAL at the top can perform some action when the modules make calls to it, and then forward these calls to the RAL beneath it. This process continues until one of the RALs finally handles the request.

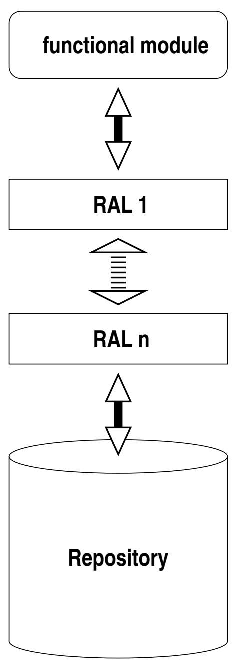

Figure 4: RALs can be stacked to add functionality

One good example where this construction makes sense is when implementing a cache. We implemented a RAL that caches all schema data in a dedicated data structure in main memory. This schema data is often very limited in size and is requested very frequently. At the same time, the schema data is the most difficult to query from a DBMS because of the transitivity of the subClassOf and subPropertyOf properties. This schema-caching RAL can be placed on top of arbitrary other RALs, handling all calls concerning schema data. The rest of the calls are forwarded to the underlying RAL.

[6] A good DBMS implementation will also cache query results to improve performance

4.3 The Repository

Thanks to the Repository Abstraction Layer, Sesame can be based on any kind of repository that is able to store RDF. The following is a list of possible concrete implementation of the repository, each with their own advantages.

- DMBS's
  Any kind of database can be used: relational databases (RDBMS), object-relational databases (ORDBMS), etc.

- Existing RDF stores
  A number of RDF stores are currently in development ([Guha, 2001, Reggiori, 2001, Beckett, 2001, Wagner, 2001]). Sesame can use such an RDF store if a RAL is written that knows how to talk to that specific RDF store.

- RDF files
  Files containing RDF can be used as repositories too. A flat file is not very practical on its own, as it will be painfully slow in storing and retrieving data. However, when combined with a RAL that caches all of the data in memory it becomes a good alternative for small volumes of data.

- RDF network services
  Apart from performance, there is no need for the repository to be located close to Sesame. Any network service that offers basic functionality for storing, retrieving and deleting RDF data can be used by Sesame. An example of a system offering such functionality is, of course, Sesame itself. Many of the RDF stores mentioned above can also be approached as Web services.

The last option in particular is very interesting. An initial query is sent to a Sesame server somewhere on the Web. This server can use not only its local repository to answer the query, but also any number of remote repositories that it knows about. In turn, some of these remote repositories might themselves either answer the query using local data-stores, or in turn again approach yet other remote repositories. This opens up the possibility of a highly distributed architecture for RDF(S) storing and querying, that has been unexplored until now, but that is truly in the spirit of the Semantic Web.

4.3.1 PostgreSQL

The first and, so far, only repository that has been used with Sesame is PostgreSQL[7]. PostgreSQL is a freely available (open source) object-relational DBMS that supports many features that normally can only be found in commercial DBMS implementations.

One of the main reasons for choosing PostgreSQL is that it is an object-relational DBMS, meaning that it supports subtable relations between its tables. As these subtable relations are also transitive, we use these to model the subsumption reasoning of RDF Schema.

The RAL that was implemented uses a dynamic database schema that was inspired by the schema shown in [Karvounarakis et al., 2000]. New tables are added to the database whenever a new class or property is added to the repository. If a class is a subclass of other classes, the table created for it will also be a subtable of the tables for the superclasses. Likewise for properties being subproperties of other properties. Instances of classes and properties are inserted as values into the appropriate tables. Figure 5 gives an impression of the contents of a database containing the data from figure 2.

The actual schema involves one more table called resources. This table contains all resources and literal values, mapped to a unique ID. These ID's are used in the tables shown in the figure to refer to the resources and literal values. The resources table is used to minimize the size of the database. It ensures that resources and literal values, which can be quite long, only occur once in the database, saving potentially large amounts of memory.

[7] See `http://www.postgresql.org/`

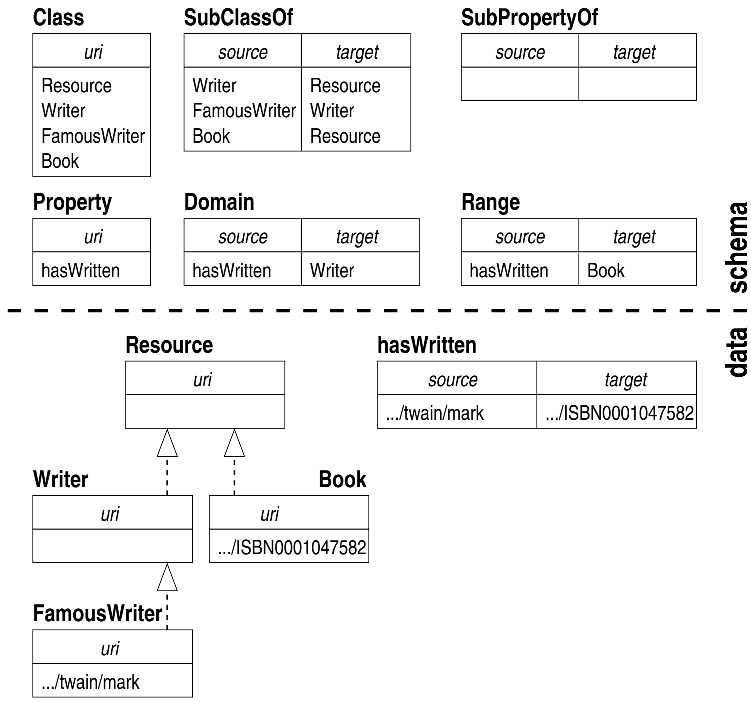

Figure 5: Impression of the object-relational schema currently used with PostgreSQL

5 Sesame's Functional Modules

In this section, we briefly describe the three modules that are currently implemented in Sesame.

5.1 The RQL Query Module

As we have seen in section 4, one of the three modules currently implemented in Sesame is an RQL query engine. RQL [Karvounarakis et al., 2000, Alexaki et al., 2000] is a proposal for a declarative language for RDF and RDF Schema. It is being developed within the European IST project C-Web and its followup project MESMUSES by the Institute of Computer Science at FORTH, in Greece.

In Sesame, a version of RQL was implemented that is slightly different from the language proposed by [Karvounarakis et al., 2000]. The Sesame version of RQL features better compliance to W3C specifications, including support for optional domain- and range restrictions as well as multiple domain- and range restrictions. See [Broekstra and Kampman, 2001] for details.

The Query Module follows the path depicted in figure 6 when handling a query. After parsing the query and building a query tree model for it, this model is fed to the query optimizer which transforms the query model into an equivalent model that will evaluate more efficiently.

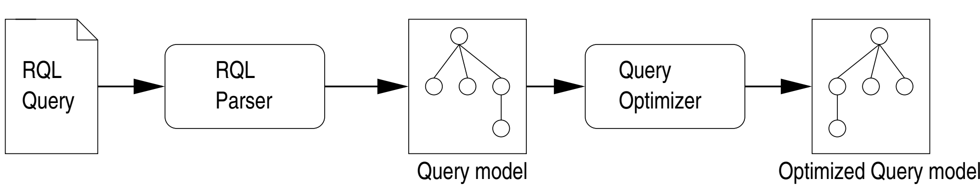

Figure 6: A query is parsed and then optimized into an query object model

The optimized model of the query is subsequently evaluated in a streaming fashion, following the tree structure into which the query has been broken down. Each object represents a basic unit in the original query and evaluates itself, fetching data from the RAL where needed. The main advantage of this approach is that results can be returned in a streaming fashion, instead of having to build up the entire result set in memory first.

In Sesame, RQL queries are translated (via the object model) into a set of calls to the RAL. This approach means that the main bulk of the actual evaluation of the RQL query is done in the RQL query engine itself.

For example, when a query contains a join operation over two subqueries, each of the subqueries is evaluated, and the join operation is then executed by the query engine on the results.

Another approach would be to directly translate as much of the RQL query as possible to a query specific for the underlying repository. An advantage of this approach is that, when using a DBMS, we would get all its sophisticated query evaluation and optimization mechanisms for free. However, a large disadvantage is that the implementation of the query engine is directly dependent on the repository being used, and the architecture would lose the ability to easily switch between repositories.

This design decision is one of the major differences between Sesame and the RDF Suite implementation of RQL by ICS-FORTH (see [Alexaki et al., 2000]). The RDF Suite implementation relies on the underlying DBMS for query optimisation. However, this dependency means that RDF Suite cannot as easily be transported to run on top of another storage engine.

A natural consequence of our choice to evaluate queries in the RAL is that we need to devise several optimization techniques in the engine, since we cannot rely on any given DBMS to do this for us.

5.2 The Admin Module

In order to be able to insert RDF data and schema information into a repository, Sesame provides an admin module. The current implementation is rather simple and offers two main functions:

1. incrementally adding RDF data/schema information;

2. clearing a repository.

The admin module retrieves its information from an RDF(S) source (usually an online RDF(S) document in XML-serialized form) and parses it using a streaming RDF parser (currently, we use the SiRPAC RDF parser [Barstow and Melnik, 2000]). The parser delivers the information to the admin module on a per-statement basis: (S, P, O). The admin subsequently checks each statement for consistency with the information already present in the repository, and infers implied information if necessary, as follows:

- if P equals type, then the admin infers that O must be a class.

- if P equals subClassOf, then the admin infers that both S and O are classes.

- if P equals subPropertyOf, then the admin infers that both S and O are properties.

- if P equals domain or range, the admin infers that S must be a property and O must be a class.

In all these cases, the admin module checks whether the inferred information is consistent with the current contents of the repository. and if so, the inferred information is added to the repository.

If the admin module encounters a duplicate statement (i.e. a fact that is already known in the repository), this is reported but otherwise ignored.

5.3 The RDF Export Module

The RDF Export Module is a very simple module. This module is able to export the contents of a repository formatted in XML-serialized RDF. The idea behind this module is that it supplies a basis for using Sesame in combination with other RDF tools, as all RDF tools will at least be able to read this format.

Some tools, like ontology editors, only need the schema part of the data. On the other hand, tools that don't support RDF Schema semantics will probably only need the non-schema part of the data. For these reasons, the RDF Export Module is able to selectively export the schema, the data, or both.

6 Experiences

Our implementation of Sesame can be found at `http://sesame.aidministrator.nl` (July 2001), and is freely available for non-commercial use. This implementation follows the general architecture described in this paper, using the following concrete implementation choices for the modules:

- As discussed above, the repository is realised by PostgreSQL.
- A protocol handler is realised using HTTP.
- The admin module uses the SiRPAC RDF parser.

In this section, we briefly report on our experiences with various aspects of this implementation.

6.1 Using RQL

As we have seen in section 5.1, Sesame supports querying using a declarative language called RQL. RQL is very powerful language that offers very expressive querying capabilities. One of the most distinguishing features of RQL is its built-in support for RDF Schema semantics and the possibility to combine data and schema information in a single query.

However, RQL currently lacks support for semantically querying *reified* statements. The reason for this is mainly that reification is poorly defined in the RDF specification. The direct result of the lack of support is that it is not possible to query such constructs semantically. When confronted with reified statements, RQL queries will have to be formulated in terms of the structure of such a statement.

6.2 Application: On-To-Knowledge

Sesame is currently being deployed as the central infrastructure of the European IST project On-To-Knowledge[8]. On-To-Knowledge aims at developing ontology-driven knowledge-management tools. Figure 7 shows how Sesame serves as the central data repository for a number of tools:

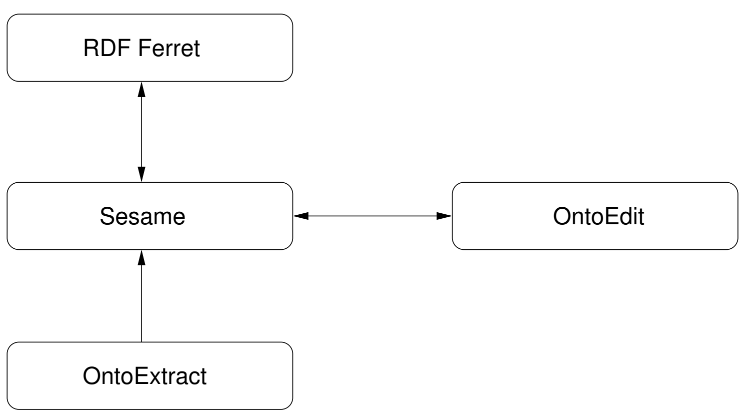

Figure 7: Sesame is positioned as a central tool in the On-To-Knowledge project

- OntoExtract, developed by CognIT a.s., extracts ontological conceptual structures from natural language documents. These ontologies are uploaded for storage in Sesame.
- The resulting ontologies can be downloaded into OntoEdit, an editor for ontologies developed by the the Institute AIFB of the University of Karlsruhe. When the user has edited an ontology, the result is again stored in Sesame.

[8] `http://www.ontoknowledge.org/`

- The resulting ontologies are downloaded into RDF Ferret, a user front-end, developed by BT Adastral Park Research Labs, that provides search and query facilities for webdata based on the ontologies.

Because Sesame is a server-based application, the integration of all this functionality is done simply by establishing HTTP connections to Sesame. We are currently in the process of applying this architecture in a number of knowledge-management applications.

6.3 Ontologies and RDF Schema

While developing Sesame, many unclarities in the RDF Schema specification were uncovered. One of the reasons for this is that RDF Schema is defined in natural language: no formal description of its semantics is given. As a result of this, the RDF Schema specification even contains some inconsistencies.

Another reason why RDF Schema is so hard to understand is that RDF Schema is self-describing in the sense that the definition of its terms is itself done in RDF Schema. This leads to strange circular dependencies in the term definitions (e.g. the term Class is both a subclass of and an instance of Resource, which is itself an instance of Class again). In fact, primitives from different meta-levels of description have been mapped to identical terms, resulting in a rather unclear specification (see also [Nejdl et al., 2000]).

One of the consequences of the circular dependencies is that RDF Schema is not only a language for, but also a part of ontologies. This means that all primitives defined in RDF Schema (i.e. subClassOf, subPropertyOf, domain, range, etc.) are also in the ontology. We would argue that this is counterintuitive. At the very least this approach deviates from approaches taken by most other ontology languages.

6.4 Using PostgreSQL

Our experiences with the database schema on PostgreSQL, as shown in section 4.3.1, are not completely satisfactory. Both data retrieval and data insertion are not as fast as we would like. Especially incremental uploads of schema data can be very slow, since table creation is very expensive in PostgreSQL. Even worse, when adding a new subClassOf relation between two existing classes, the complete class hierarchy starting from the subclass needs to broken down and must then be rebuilt again because subtable relations can not be added to an existing table; the subtable relations have to be specified when a table is created. Once created, the subtable relations are fixed.

6.5 Scalability issues

We have been experimenting with several data sets and/or ontologies that are currently available on the Web. The largest set of data that we have uploaded and subsequently queried was the collection of nouns from Wordnet[9], consisting of about 400,000 RDF statements. This data set almost completely exists of RDF data (i.e. hardly any schema information). While we have not performed any structured benchmark testing, the following points are noteworthy.

First of all, all experimenting has been done using a desktop computer (Sun UltraSPARC 5 workstation, 256MB) to run Sesame. Java Servlets running on a web server were used as protocol handlers to communicate over HTTP. The database schema described in section 4.3.1 in combination with PostgreSQL version 7.1.1 was used as repository.

The uploading of the information is not as fast as we would like, mainly due to the database schema being used. Just adding a data statement to the database involves doing the following steps:

- Check whether the property is already known. If not, add it and create a table for it.
- Check whether the subject is already known, adding it if not.
- Check whether the object is already known, adding it if not.
- Add a row representing the statement to the appropriate table.

[9]This collection can be found in RDF form at `http://www.semanticweb.org/library/`

Most of these steps have to be performed in sequential order, which is very time-intensive. Uploading the Wordnet nouns took approximately 94 minutes, which comes down to 71 statements per second. As was to be expected, the upload did not show any significant signs of slowing down as the amount of data in the repository increased (the amount of data is really not very large by DBMS standards).

Querying the information proved to be quite slow too, for exactly the same reasons. Due to the distributed storage over multiple tables, retrieving data from the repository means doing many joins on tables, hindering performance.

7 Future directions

7.1 DAML+OIL

Currently, Sesame understands the semantics of RDF and RDF Schema. We would like to extend this to more powerful languages like DAML+OIL [Horrocks et al., 2001]. DAML+OIL is an extension of RDF Schema and offers additional primitives for creating schemata. Examples of the additional expressive power of DAML+OIL are:

- the use of arbitrary class expressions, including disjunction, conjunction and negation (complement) of classes
- cardinality constraints on properties, expressing the minimal and maximal number of values a property can have for each object
- symmetric, transitive and inverse properties

Since DAML+OIL allows more expressiveness and has more inferencing capabilities, a reasoner/query language that understands its semantics is significantly more complicated.

7.2 Other repositories

We are planning to implement support for other DMBS/schema combinations so that we can compare the pros and cons of each approach. A first option will be to implement a RAL based on a traditional relational DBMS, i.e. one that only uses standard SQL queries. Such a RAL can be used on lots of DBMS's as almost all DBMS's support these queries.

7.3 Admin Module

The Admin Module currently offers very limited functionality for administrating the contents of repositories. More fine-grained functionality is needed for the module to be really useful. We are currently investigating the options to accomplish this. One of the options is to extend RQL with primitives for updating and deleting data, just like SQL does.

8 Conclusion

In this paper we have presented Sesame, a flexible architecture for storing and querying both RDF data and RDF Schema information. Sesame is an important step beyond the currently available storage and query devices for RDF, since it is the first publicly available implementation of a query language that is aware of the RDF Schema semantics.

An important feature of the Sesame architecture is its abstraction from the details of any particular repository used for the actual storage. This makes it possible to port Sesame to a large variety of different repositories, including relational databases, RDF triple stores, and even remote storage services on the Web.

Sesame itself is a server-based application, and can therefore be used as a remote service for storing and querying data on the Semantic Web. As with the storage layer, Sesame abstracts from any particular communication protocol, so that Sesame can easily be connected to different clients by writing different protocol handlers.

We have constructed a concrete implementation of the generic architecture, using PostgreSQL as a repository and using HTTP as communication protocol handlers.

Important next steps to expand Sesame towards a full fledged storage and querying service for the Semantic Web include its extension from RDF Schema to DAML+OIL and implementations for different repositories, notably those that can live elsewhere on the Web.

References

[Alexaki et al., 2000] Alexaki, S., Christophides, V., Karvounarakis, G., Plexousakis, D., and Tolle, K. (2000). The RDFSuite: Managing Voluminous RDF Description Bases. Technical report, Institute of Computer Science, FORTH, Heraklion, Greece. See `http://www.ics.forth.gr/proj/isst/RDF/RSSDB/rdfsuite.pdf`.

[Barstow and Melnik, 2000] Barstow, A. and Melnik, S. (2000). SiRPAC: Simple RDF Parser and Compiler. `http://www.w3.org/RDF/Implementations/SiRPAC/`.

[Beckett, 2001] Beckett, D. (2001). Redland RDF Application Framework. `http://www.redland.opensource.ac.uk/`.

[Box et al., 2000] Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H. F., Thatte, S., and Winer, D. (2000). Simple Object Access Protocol (SOAP) 1.1. W3c note, World Wide Web Consortium. See `http://www.w3.org/TR/SOAP/`.

[Brickley and Guha, 2000] Brickley, D. and Guha, R. (2000). Resource Description Framework (RDF) Schema Specification 1.0. Candidate recommendation, World Wide Web Consortium. See `http://www.w3.org/TR/2000/CR-rdf-schema-20000327`.

[Broekstra and Kampman, 2001] Broekstra, J. and Kampman, A. (2001). Query Language Definition. On-To-Knowledge (IST-1999-10132) Deliverable 9, Aidministrator Nederland b.v. See `http://www.ontoknowledge.org/`.

[Cattel et al., 2000] Cattel, R., Barry, D., Berler, M., Eastman, J., Jordan, D., Russell, C., Schadow, O., Stanienda, T., and Velez, F. (2000). *The Object Database Standard: ODMG 3.0*. Morgan Kaufmann.

[Chamberlin et al., 2001] Chamberlin, D., Florescu, D., Robie, J., Simeon, J., and Stefanescu, M. (2001). XQuery: A Query Language for XML. Working draft, World Wide Web Consortium. See `http://www.w3.org/TR/xquery/`.

[Fensel et al., 2000] Fensel, D., van Harmelen, F., Klein, M., Akkermans, H., Broekstra, J., Fluit, C., van der Meer, J., Schnurr, H.-P., Studer, R., Hughes, J., Krohn, U., Davies, J., Engels, R., Bremdal, B., Ygge, F., Lau, T., Novotny, B., Reimer, U., and Horrocks, I. (2000). On-To-Knowledge: Ontology-based Tools for Knowledge Management. In *Proceedings of the eBusiness and eWork 2000 (EMMSEC 2000) Conference*, Madrid, Spain.

[Guha, 2001] Guha, R. (2001). rdfDB. `http://web1.guha.com/rdfdb/`.

[Horrocks et al., 2001] Horrocks, I., van Harmelen, F., Patel-Schneider, P., Berners-Lee, T., Brickley, D., Connoly, D., Dean, M., Decker, S., Fensel, D., Hayes, P., Heflin, J., Hendler, J., Lassila, O., McGuinness, D., and Stein, L. A. (2001). DAML+OIL. `http://www.daml.org/2001/03/daml+oil-index.html`.

[Karvounarakis et al., 2000] Karvounarakis, G., Christophides, V., Plexousakis, D., and Alexaki, S. (2000). Querying community web portals. Technical report, Institute of Computer Science, FORTH, Heraklion, Greece. See `http://www.ics.forth.gr/proj/isst/RDF/RQL/rql.pdf`.

[Lassila and Swick, 1999] Lassila, O. and Swick, R. R. (1999). Resource Description Framework (RDF): Model and Syntax Specification. Recommendation, World Wide Web Consortium. See `http://www.w3.org/TR/REC-rdf-syntax/`.

[Melnik, 2000] Melnik, S. (2000). RDF API Draft. public draft, Database Group, Stanford University. See `http://www-db.stanford.edu/~melnik/rdf/api.html`.

[Nejdl et al., 2000] Nejdl, W., Wolpers, M., and Capella, C. (2000). The RDF Schema Revisited. In *Modelle und Modellierungssprachen in Informatik und Wirtschaftsinformatik, Modellierung 2000, St. Goar*. Foelbach Verlag, Koblenz.

[Reggiori, 2001] Reggiori, A. (2001). RDFStore. `http://rdfstore.jrc.it/`.

[Wagner, 2001] Wagner, H. (2001). Extensible open rdf. `http://eor.dublincore.org/`.