Foundation for Research and Technology Hellas
Institute of Computer Science
Information Systems Laboratory

ONTOLOGY STORAGE AND QUERYING

TECHNICAL REPORT No 308
April 2002

Aimilia Magkanaraki*, Grigoris Karvounarakis*, Ta Tuan Anh,
Vassilis Christophides*, Dimitris Plexousakis*
* {aimilia, gregkar,christop, dp}@ics.forth.gr
ta@enst.fr

1. Introduction

The necessity for ontology building, annotating, integrating and learning tools is uncontested. However, the sole representation of knowledge and information is not enough. Human information consumers and web agents have to use and query ontologies and the resources committed to them, thus the need for ontology storage and querying tools arises. However, the context of storing and querying knowledge has changed due to the wide acceptance and use of the Web as a platform for communicating knowledge. New languages for querying (meta)data based on web standards (e.g., XML[23], RDF[17], Topic Maps [16]) have emerged to enable the acquisition of knowledge from dispersed information sources, while the traditional database storage techniques have been adapted to deal with the peculiarities of the (semi)structured data on the web.

The purpose of this chapter is to briefly present and evaluate a set of query languages and associated tools for ontology/resource storage and querying aiming to support large-scale Semantic Web [3] applications, the next evolution step of the Web. This list of languages and tools is by no means complete and the tools presented are indicative of the tendency to provide full storage and query support to web-based ontology/metadata standards, such as RDF [17], RDFS [4], Topic Maps [16], DAML+OIL [9] or the forthcoming Web Ontology Language [15]. Our work in this chapter focuses on the evaluation of querying languages designed for Semantic Web related knowledge representation formalisms rather than general-purpose querying languages (e.g., SQL, Datalog, F-logic). Although it has to be proven in practice, RDF-enabled search technologies have the potential to provide a significant improvement over the current keyword-based engines or theme navigation search, especially when it comes to conceptual browsing and querying [19]. Furthermore, this orientation facilitates the comparison of querying languages and tools, since it provides a common reference base.

It should be stressed that our comparison of ontology query languages and tools does not rely on performance figures, since these would require extensive comparative experiments, which go beyond the scope of this work. On the contrary, we present an overview of general system features and query language expressiveness, while providing the interested reader with useful references to additional informative material. Section 2 presents the evaluation framework we have employed for our comparison, while Section 3 hosts a short description of the query languages and storage/query tools selected for our survey. Section 4 provides the comparison of the query languages' expressive power, as well as the technical characteristics of the related tools. This was considered quite useful in order to separate the theoretical foundations underlying a query language from its practical deployments in various tools. Finally, Section 5 concludes our survey.

2. Evaluation framework of Query Languages and Storage Tools

The purpose of this section is to present the evaluation framework we have adopted for comparing, on the one hand, the expressiveness of the query languages as opposed to the underlying data model and, on the other hand, the technical features of the supporting tools. Before presenting our comparison according to these evaluation axes, we briefly present each query language and tool in terms of general descriptive criteria. In particular, for both cases, we present a brief general description, some **references** from which an interested reader could acquire more information and a **URL** for additional informative material. Especially for the description of the storage and querying tools, we also provide the web pages from where the interested reader could find **documentation** material, **tutorials** about the tool or on-line **demonstrations** of the features supported by the tool. Furthermore, we provide information about the current release (**version**) and the software **platform** of the tool, as well as the **pricing policy** followed by the tool developers.

A general evaluation framework used to compare the ontology query languages comprises basic criteria for describing the language characteristics, namely the ontology/metadata **standard** for which the language has been proposed, the **data model** used for capturing the generated description bases and ontologies and the **language of origin** on which the query language has been based. Moreover, we compare the query languages on their **closure**, i.e., the ability of the language to support functional composition of queries, on their **orthogonality**, which indicates whether the language permits any kind of data as input and output of queries and their **generality**, i.e., whether the language exploits all the

primitives of the ontology/metadata model. This last criterion can form the basis of a more detailed query language evaluation framework. However, due to the fact that the query languages for Topic Maps [16] and DAML+OIL [9] are still in a preliminary phase, we focus our detailed comparison on query languages for RDF/S ([17], [4]).

The criteria constituting the framework for evaluating the expressive power of RDF/S ontology query languages are distinguished in five categories, namely criteria regarding Modeling Constructs, Ontology Querying, Data Querying, Data/Ontology Querying and Additional Features. The criteria in the category ***Modeling Constructs*** refer to the ability of the query language to support the basic modeling constructs of the underlying standard (RDF/S). More specifically, we record whether a specific query language can perform queries on classes, properties and resources coming from different **namespaces/multiple schemas** and whether it supports **concrete data types**, **container values** and the modeling mechanisms of **multiple inheritance**/ **instantiation** and **reification**. Multiple inheritance permits the declaration of a class (property) as subclass (sub-property) of many classes (properties), while reification provides mechanisms for modeling statements about statements. Instantiation is a mechanism for defining a resource as instance of one or more classes/properties.

In the category ***Ontology Querying*** there are criteria for stating whether the query language can exploit the presence of ontology (schema) knowledge, i.e., if the **ancestor/descendant traversal of class/property hierarchies** can be performed and what **filtering conditions** can be posed **on class/property hierarchies**. The basic criterion of the ***Data Querying*** is the ability of the query language to provide constructs for calculating the **extent of a property/class**, i.e., the ability to find all the resources defined as instances of a particular property/class. Furthermore, in this category fall criteria such as the support of **complete Boolean filters** (negation, conjunction, disjunction), **set-based operations** (union, intersection, difference), **arithmetic operations** on data values and **container values constructors**, i.e., language constructs that can be used to build sequences or bags. The category ***Data/Ontology Querying*** contains criteria referring to the competence of the query language in combining data and ontology querying in a query expression. The basic way to perform such an advanced query is to use **generalized path expressions**. A generalized path expression queries data and ontology at the same time, thus enabling the formulation of queries on description bases without exact knowledge of the ontologies employed. Generalized path expressions, as well as the support of **existential/universal quantifiers** and **nested queries** are indicative criteria of the expressive power of a query language. Lastly, the category **Additional Features** hosts a number of criteria that can be used to evaluate the effectiveness of the query language and the added value of its use when it comes to real, large scale applications. The support of **aggregate** (min, max, average, sum, count), **grouping** (e.g., an SQL-equivalent to group_by clause), **sorting** (for ordering querying results) and **built-in data functions** (e.g., math and string/date converting functions), in combination with the facilities to support the definition of **arbitrary functions** and **user-defined inference rules** are features that add to the expressive power of the query language. To this last category we can also include the existence of **view definition primitives**, i.e., the facility for the user to define views over the description bases.

As far as the evaluation of the ontology storage and querying tools is concerned, we adopt an evaluation framework, which can provide an overview of the most important technical features supported by the tools presented. The criteria falling into this evaluation framework are focused on the technical characteristics of the tools, i.e., the supported **Query language**, the **Implementation language** the developers have used for deploying the tool and the **Storage database** system used to store the ontology/data. The criterion of **Inference support** indicates whether the tool permits arbitrary deduction rules for inferring new knowledge, while the criterion of **Update support** signifies the capability of the tool to modify the contents of existing ontology schema and (meta)data. A criterion that can be used to judge the extensibility and the ability of the tool to collaborate with other applications is the **API support (querying and updating)**, i.e., the ability for interfacing with clients, while the **Export data format** indicates the formats supported for exporting the ontology. Lastly, we can perform a preliminary comparison of the tools on terms of their **Scalability/Performance**. The data stated for this criterion is taken from a survey of RDF/Triple data stores by W3C[1]. More detailed data on scalability and performance would require extensive comparative testing, which go beyond the scope of our survey.

[1] http://www.w3.org/2001/05/rdf-ds/DataStore

3. Description of ontology query languages and tools

The support of querying facilities has always been a primary requirement for repositories of any kind. The proliferation of knowledge caused by the widespread use of the Web as a knowledge communication platform has posed the same and even more imperative requirements for performing queries and thus locating desirable resources into the vast information space. However, the data models used to represent and encode knowledge on the Web differ from the traditional data structures. RDF [17], RDFS [4], DAML+OIL [9] and Topic Map [16] are the emerging standards used to encode web-based data. Thus, the functionality a querying language should support must take into account the structure and the peculiarities of the new paradigms. In particular, RDF and DAML+OIL are based on a directed labelled graph data model, which allows labels both on nodes and edges, while XML [23] uses labels only on edges. Both models can be serialized in a number of different representations, one of which is the use of triples in the form of <predicate, subject, object>. Topic Maps can also be expressed using XML. One difference between RDF and Topic Map is that the latter is centred on topics while the former on resources. Furthermore, while RDF annotates directly the resources, Topic Map creates a semantic network layer - a "virtual map"- above the information resources, thus leaving the information resources unchanged.

In this section we will focus on query languages for RDF, DAML+OIL and Topic Maps. The reason is that RDF, with the aid of RDF Schema Language [4], provides a rich infrastructure for representing meaningful information in the form of ontologies. Topic Maps, on the other hand, provides mechanisms for creating user-editable views of heterogeneous information repositories, while DAML+OIL extends RDF/S with richer modelling primitives commonly found in frame-based languages. Since for a query language to be deployed there must exist storage tools, we also present a set of ontology storage tools providing querying capabilities, as well as inference engines attached to them.

3.1 Ontology Query Languages

3.1.1 ICS-FORTH RQL

RQL, developed in the context of the EU projects C-Web (IST-1999-13479) and MesMuses (IST-2001- 26074), is a typed, declarative query language for querying RDF description bases following a functional approach a la OQL. It is defined by a set of basic queries and iterators, which can be used to build new ones through functional decomposition. RQL relies on a formal graph model that enables the interpretation of superimposed resource descriptions by representing properties as self-existent individuals and introducing a graph instantiation mechanism that permits multiple classification of resources. It adapts the functionality of semi-structured or XML query languages to the peculiarities of RDF (i.e., labels on both graph nodes and edges, taxonomies of labels) but, foremost, it extends this functionality by uniformly querying both resource descriptions and (meta)schemas. In particular, the novelty of RQL lies in its ability to smoothly combine ontology and data querying while exploiting - in a transparent way - the taxonomies of labels and multiple classification of resources. Thus, users are able to query resources described according to their preferred ontology, while discovering in the sequel how the same resources are also described using another classification ontology (schema). RQL can compose schema paths to perform more complex ontology navigation, a kind of query not expressible in existing languages with ontology querying capabilities, while it supports generalized path expressions featuring variables on both labels for nodes (i.e., classes) and edges (i.e., properties). Furthermore, it features set-based queries and supports XML Schema data types, grouping primitives, aggregate functions and arithmetic operations on data values.
URL: http://139.91.183.30:9090/RDF/RQL/index.html
References: - Gregory Karvounarakis, Sofia Alexaki, Vassilis Christophides, Dimitris Plexousakis, Michel Scholl. "*RQL: A Declarative Query Language for RDF"*. WWW2002, May 7-11, 2002, Honolulu, Hawaii, USA. ACM 1-58116-449-5/02/0005
- Grigoris Karvounarakis, Vassilis Christophides. "*The RQL v1.5 User Manual*".
<http://139.91.183.30:9090/RDF/RQL/Manual.html>

3.1.2 ILRT SquishQL

SquishQL constitutes the proposition of the ILRT[2] Semantic Web Research Group to the need of a query language for RDF description bases. Being a simple graph-navigation query language for RDF based on a subgraph matching mechanism, SquishQL uses SQL-like constructs to reflect RDF's graph syntax. Apart from supporting a query model based on a graph pattern formed from variables for nodes, arcs and literals, it introduces filter functions in the form of Boolean expressions over the variables. Thus, in SquishQL there are two classes of constraints: patterns and filter expressions. The pattern language is formed from triple patterns <subject, predicate, object> describing edges of the graph and a conjunction operator. For each component of a triple, i.e., subject, predicate and object it allows either a variable or an explicit value. Filter functions restrict the values that the variables over the components of a triple can take. In general, the patterns are generative, since they create bindings and the filters are restrictive, in view of the fact that they remove possibilities. In particular, the WHERE clause corresponds to the generative part of an SquishQL query by specifying the graph pattern as a list of triple patterns and the AND clause corresponds to the restrictive part, which specifies the Boolean expressions.

SquishQL supports a considerable functionality for RDF query expression and has formed the basis of a number of RDF query languages of diverse complexity. A language derived from SquishQL is RDQL, which is being developed by the Hewlett Packard Semantic Web Group [21]. RDQL is a syntax and query API whose purpose is to act as a model-level access mechanism that is of a higher level than an RDF API. It extracts information from an RDF model by treating RDF as data and providing query with triple patterns and constraints over a single RDF model. However, SquishQL does not provide an explicit syntax for reification. Furthermore, it does not express transitivity or other forms of unknown length paths in the graph and does not support disjunction or repetition.
URL: http://ilrt.org/discovery/2001/02/squish/
References: Libby Miller, Andy Seaborne, Alberto Reggiori. "*Three Implementations of SquishQL, a Simple RDF Query Language*". To appear in the 1st International Semantic Web Conference (ISWC2002), June 9-12, 2002. Sardinia, Italy.

3.1.3 Intellidimension RDFQL

Developed by Intellidimension, RDFQL is an SQL-style, statement-based query language that supports the RDF model and syntax. In addition to providing facilities for querying RDF structured data, an important feature of RDFQL is its ability to infer new statements from existing ones by using user-defined inference rules of the form "*if A is the case then so is B*" for powerful deductive searches (rules are themselves stored as RDF statements with a table). In particular, the CREATE RULE command creates a new inference rule in a datasource, its IN clause specifies the datasource, the INFER clause specifies the head of the rule -the fact to be inferred if the correct conditions are met and the FROM clause specifies the body of the rule -the set of conditions that must be met in order for the head to be true. As an SQL-style language, it also uses commands such as INSERT, DELETE and SELECT to perform query and inference operations on RDF triples, as well as data definition commands like CREATE TABLE or CREATE VIEW. RDFQL views, which can be used in data reporting packages such as Crystal Reports, are pre-built queries that can be used to present RDF tables as a traditional tabular data structure. More specifically, the SELECT command is very similar to its SQL counterpart, with the differences that RDFQL utilizes the USING clause, instead of the FROM clause, to specify the data sources to use (similar to a table in SQL) and that field names in the select list are replaced by variables (e.g. ?predicate, ?myvar). RDFQL also includes most of the SQL functions and operators to support complex query conditions, which are defined in the WHERE clause. As an optional facility, the ORDER BY clause provides ways to order the results in an order direction (ascending or descending). The INSERT command, on the other hand, inserts a list of statements into a data source and consists of the INTO, USING and WHERE clauses. The INTO clause specifies the destination data source, the USING clause specifies a data source from which to copy statements and the WHERE clause defines any conditions that should be met by the statements being copied. Furthermore, RDFQL supports aggregation functions (Count), comparison operators (e.g., =, <, >) and aliases for each namespace using the NAMESPACE command.
URL: http://www.intellidimension.com/RDFGateway/Docs/rdfqlgettingstarted.asp
References: *RDFQL Reference Manual*
<http://www.intellidimension.com/RDFGateway/Docs/rdfqlmanual.asp>

[2] ILRT: Institute for Learning and Research Technology, University of Bristol (http://ilrt.org/).

3.1.4 RDFPath

Although still under development by the RDFPath group and other interested parties, RDFPath constitutes a language - similar to XPath [8] for XML - for localizing information encoded in an RDF graph. Its purpose is to provide general techniques for specifying paths between two arbitrary nodes of an RDF graph. In general, an RDFPath expression is a composition of a primary selection and several location steps and filters. Primary selections, localization steps and filters are RDFPath language constructs used to formulate queries against RDF graphs. In particular, a *primary selection* selects an initial set of nodes of a given RDF graph, e.g., the *resource()* and *literal()* constructs select all resources and literals of an RDF graph respectively, a *location step* specifies a set of nodes that can be reached by "one step" from a given context object, e.g., the *child()* clause selects the children of a context node and a *filter* selects a subset of a given set of objects. Thus, every primary selection or location step selects a set of objects relative to the current context and the role of the RDFPath filters is to refine these sets of objects. RDFPath is mainly focused on RDF graphs, but among the intentions of the RDFPath group is to provide compliance with the RDF Schema and support the class/property subsumption relations of the RDF Schema specification [4], i.e., the *subClassOf* and *subPropertyOf* relations.
URL: http://zoe.mathematik.uni-osnabrueck.de/RDFPath/
References: - Stefan Kokkelink, "*Transforming RDF with RDFPath*". Working Draft. March 2001.
<http://zoe.mathematik.uni-osnabrueck.de/QAT/Transform/RDFTransform.pdf>

3.1.5 VERSA RDF Query Language

Versa is an evolving, graph-based language for querying RDF models (sets of statements), initially developed by the implementers of 4RDF[3]. Its main features include traversal of arcs, processing of node contents and general expression evaluation. In fact, (transitive) traversal expressions are a core Versa feature, since they allow to match patterns in the model by respecting their structure as directed graphs. Versa also supports forward and backward traversal. In a forward traversal, we move along the specified arcs from subjects to objects, and then filtering the result, while in a backward traversal, we move from objects to subjects, and then filtering the subjects. In any case, the nature of the query results depends on the underlying RDF implementation, which determines what is considered to be a resource for Versa purposes. For instance, in the presence of an RDF schema, the object of a statement might be treated as a resource, in case the predicate of the statement is defined with a resource range. Generally, in order to support the functionality needed, Versa provides built-in functions, e.g., the all() function retrieves all the resources in the model. Due to its heavy use of functions, Versa sometimes has a LISP-like feel. Versa also provides facilities -such as boolean logic and set operations, aggregates, substring matching, and other core data type manipulation. Furthermore, for ease of expressing queries, it permits an abbreviated form for expressing resources, which consists of the namespace prefix and the latter part of the URI joined by a colon. The quester can also form a query expressed with more sophisticated criteria, e.g., to use all()-rdfs:label-> contains("ab"), if he/she wants all resources whose label contains the string "ab", as well as request the presentation of query results in a sorted way.
URL:
http://uche.ogbuji.net:8080/uche.ogbuji.net/tech/rdf/versa/versa.doc?xslt=/ftss/data/docbook_html1.xslt
References: Uche Ogbuji. "*Versa by example*".
<http://uche.ogbuji.net:8080/uche.ogbuji.net/tech/rdf/versa/versa-by-example.txt>

3.1.6 TRIPLE

TRIPLE language is an RDF query, inference, and transformation language, developed as a joint work by Stefan Decker (Stanford University Database Group) and Michael Sintek (DFKI GmbH Kaiserslautern, Knowledge Management Department and Stanford University Database Group). TRIPLE's layered and modular nature, based on Horn Logic and F-Logic, aims to support applications in need of RDF reasoning and transformation, i.e., to provide mechanisms to query web resources in a declarative way. However, contrary to many other RDF query languages, TRIPLE allows the semantics of languages on top of RDF, such as RDF Schema [4] and Topic Maps [16], to be defined with rules, instead of supporting the same functionality with built-in semantics. Wherever the definition of language semantics is not easily possible with rules (e.g., DAML+OIL [9]), TRIPLE provides access to

[3] http://www.xml.com/pub/a/2000/10/11/rdf/index.html

external programs, like description logics classifiers. Thus, two different kinds of layers are supported: syntactical extensions of Horn Logic to support basic RDF constructs, like resources and statements, and modules for semantic extensions of RDF, like RDF Schema [4], OIL [18] and DAML+OIL [9], implemented either directly in TRIPLE or via interaction with external reasoning components, such as a DL classifier. In particular, TRIPLE provides native support for resources and namespaces, abbreviations (e.g., isa:=rdf:SubClassOf), models (sets of RDF statements), reification and rules with expressive bodies (full First Order Logic syntax). TRIPLE also allows Skolem functions, which, when used in rules, can be used to transform one or several models (i.e., a set of RDF statements) into a new one, a functionality especially useful for ontology mapping or integration. Furthermore, instead of subject, predicate or object definitions, TRIPLE permits the usage of path expressions. For example, we can define (horn) rules that search for documents with a specified subject. TRIPLE provides a human-readable ASCII-syntax, as well as an RDF-based syntax for exchanging queries and rules, e.g., between communicating agents.
URL: http://www.dfki.uni-kl.de/frodo/triple/
References: Michael Sintek, Stefan Decker. "*TRIPLE-An RDF Query, Inference, and Transformation Language*". In Proceedings of the Deductive Databases and Knowledge Management Workshop (DDLP' 2001). Japan, October 2001.

3.1.7 DAML+OIL Query Language

DAML+OIL [9] is a language elaborated on top of RDF/RDFS for expressing more sophisticated classifications and properties of resources than RDFS. It provides modeling primitives commonly found in frame-based languages while its formal semantics is defined in description logics. However, a query language for the (meta)data expressed in DAML+OIL is still an ongoing work. For example, DAML-S (for "DAML Search engine") is an engine that enables querying a DAML ontology. The form of query language used is "FIND … SUCH-THAT …END", that allows finding resources satisfying a conjunction of statements (triples). We can also find a discussion about the DQL language (DAML+OIL Query Language) in the archives of joint-committee@daml.org email list. Although DQL is still under development, it is defined with two parts, a Query Premise and a Query Pattern. The Query Premise is a condition checking on the queried KB. A query premise seems important in that it allows a query to hypothesize an object (e.g., "if Foo is a Person with two male siblings …") and then make questions about that hypothesized object. The Query Pattern corresponds to the "from" clause in RQL [section 3.1.1], but there is nothing in DQL corresponding to the RQL "select" and "where" clause. Continuously, in an internet paper, Arnold deVos had proposed an RDF query language based on DAML+OIL which allows to query both RDF(S) and DAML+OIL (meta)data. A query in this language is formulated with an expression of the form "select … from …", while the query results are only the triples. Nevertheless, the "from" clause, which is an expression describing a DAML class (where the statements are found), can be used for expressing DAML+OIL complex concept properties. Finally, a query language for DAML+OIL proposed by the University of Manchester [12] also allows exploiting DL systems in order to provide complete reasoning services. This language once again relies on a triple data model. While concrete data types have been introduced, a query in this language representing a boolean conjunction of statements can return a true/false answer or a set of arbitrary tuples.
URL: http://www.daml.org/listarchive/joint-committee/0665.html,
http://www.csl.sri.com/papers/denkeretal01/
References: - Arnold deVos. "*An RDF query language based on DAML*"
<http://www.langdale.com.au/RDF/DAML-Query.html>
- Grit Denker, Jerry R. Hobbs, David Martin, Srini Narayanan, Richard Waldinger, "*Accessing Information and Services on the DAML-Enabled Web*", SRI International Menlo Park, California.
<http://www.ai.sri.com/daml/notes/HW2/SemanticWeb/paper.html>

3.1.8 Topic Maps Query Language

Topic Maps Query Language (TMQL) is a standardization project of ISO (JTC1 SC34 WG3) and topicmaps.org, the organizations who contributed to the development of ISO Topic Maps [16] and XTM (XML Topic Maps) [20]. Although at requirements phase, TQML aspires to constitute the SQL equivalent for Topic Maps, i.e., an XML-based query language suited to satisfy the data access requirements of Topic Maps (TMs). TMQL is intended to use SQL-ish constructs familiar to most developers, so as to simplify its use. However, the context of use for SQL and TMQL are different: SQL was developed for use in relational database environments, where data have well-defined structure. On the other hand, TMQL must be applicable to environments of vast amount, semi-

structured and constantly changing information. Thus, the semantics of expressing a typical SQL select query (used to retrieve data from a table in a database) must be adapted to retrieving data from a Topic Map. One of the requirements posed for TMQL is the conformance with the reference data model, as well as the support of operations on the information content of a topic map, i.e., adding/ removing/ retrieving information and manipulation of information integrity constraints.
URL: http://www.garshol.priv.no/download/tmlinks.html,
http://groups.yahoo.com/group/tmql-wg/files/official-docs/tmqlreqs.html
References: Rafal Ksiezyk. "*Answer is just a question [of matching Topic Maps]*". In Proceedings of XML Europe 2000, 12-16 June 2000. Palais de Congres, Paris, France.
<http://www.gca.org/papers/xmleurope2000/papers/s22-03.html>

3.1.9 Ontopia Tolog

Proposed and deployed by Ontopia[4], *tolog* combines elements from Prolog and SQL to query topic maps bases. The language consists of predicates, which can be thought of as tables of values. In this context, queries are executed by matching them against the predicates and binding logical variables in the queries to the values produced by the predicates. Tolog supports two kinds of predicates: built-in predicates and association predicates, which are like the built-in ones except that one must explicitly define the association roles used in the query. The built-in predicates, namely *instance-of(instance,class)*, *direct-instance-of(instance,class)* and *a/=b* (for comparing two values) are part of the language definition and can be used as queries themselves. Apart from querying instance-of relationships and associations, tolog supports projection of query results (that is, leave out variables only used for intermediate computations), counting of query matches, ordering of query results according to specific variables, a form of negation (the not operator is used as a filter) and alternative sets of predicates in computations that can be used to query more than one alternative path through a topic map in a single query. Furthermore, inference rules can be used to infer associations not explicitly present in the topic map. The addition of inference rules to tolog enables the query language to perform logical inference to deduce new facts that are implied by the information already in the topic map, even if they are not explicitly stated anywhere. Rules, in the form of [*head(predicates):-body of the rule*], can be kept in special rules files, which are attached to the topic map as topic map metadata, or they can be fed to the query processor whenever they are wanted or needed.
URL: http://www.ontopia.net/omnigator/docs/query/tutorial.html
References: - Lars Marius Garshol. "tolog. A topic map query language". In Proceedings of XML Europe 2001, 21-25 May 2001, Berlin, Germany.
- "*Tolog: Language Tutorial"*. Version 1.3.
<http://www.ontopia.net/omnigator/docs/query/tutorial.html>

3.2 Ontology Storing and Querying Tools

3.2.1 ICS-FORTH RDFSuite

The ICS-FORTH RDFSuite, partially supported by EU projects C-Web (IST-1999-13479) and MesMuses (IST-2001- 26074), is a suite of tools for RDF metadata management, addressing the need of RDF metadata processing for large-scale Web-based applications. It consists of tools for parsing, validating, storing and querying RDF descriptions, namely the Validating RDF Parser (VRP), the RDF Schema Specific DataBase (RSSDB) and the RDF Query Language (RQL). RSSDB is a persistent tool for loading resource descriptions in an object-relational DBMS (e.g., PostgresSql) by exploiting the available RDF schema knowledge. It preserves the flexibility of RDF in refining schemas and/or enriching descriptions at any time whilst it can be customized in several ways (as opposed to triple-based repositories) according to the specificities of both the manipulated RDF descriptions (i.e., schemas) and the underlying RDF application queries. Its main goal is the separation of RDF schema information from data information, as well as the distinction between unary and binary relations holding the instances of classes and properties. Querying of stored RDF descriptions is accomplished by the query module, which implements the RQL language [section 3.1.1]. For performance reasons, the module pushes as much as possible query evaluation to the underlying DBMS, while benefiting from robust SQL3 query engines and DB indices. The RQL module is easy to integrate with web application servers and it is easy to couple with other commercial ORDBMS.

[4] http://www.ontopia.net/

URL: http://139.91.183.30:9090/RDF/index.html
References: -S. Alexaki, V. Christophides, G. Karvounarakis, D. Plexousakis, K. Tolle. "*The ICS-FORTH RDFSuite: Managing Voluminous RDF Description Bases*". 2nd International Workshop on the Semantic Web (SemWeb'01), in conjunction with Tenth International World Wide Web Conference (WWW10), pp. 1-13, Hong Kong. May 1, 2001.
- S. Alexaki, V. Christophides, G. Karvounarakis, D. Plexousakis. "*On Storing Voluminous RDF Descriptions: The case of Web Portal Catalogs*". In Proceedings of the 4th International Workshop on the Web and Databases (WebDB'01)- In conjunction with ACM SIGMOD/PODS, Santa Barbara, CA. May 24-25, 2001.
Documentation: http://athena.ics.forth.gr:9090/RDF/RQL/Design.html
Tutorial: http://139.91.183.30:9090/RDF/RQL/Manual.html
Version: 1.5
Platform: Sun Solaris and Linux
Demonstration: available at http://139.91.183.30:8999/RQLdemo/
Pricing Policy: open source/free software under RDFSuite License (GPL compatible).

3.2.2 Sesame

Sesame, an RDF Schema-based Repository and querying facility, is being developed by Aidministrator Nederland bv as one of the key deliverables in the European IST project On-To-Knowledge (EU-IST-1999-10132). It is a system consisting of a repository, a query engine and an administration module for adding and deleting RDF data and Schema information. It supports expressive querying of RDF data and schema (ontology) information, using the RQL query language [section 3.1.1] and understands the semantics of most of the RDF Schema classes and properties. Thus, it supports the basic inferencing needed for supporting RDF Schema, such as transitivity of subClassOf- and subPropertyOf-properties. The RQL implementation of Sesame is slightly different from the ICS-FORTH RDFSuite's, since the interpretation of the RDF Schema differs in the two cases and the Sesame's query engine does not support all features of RQL. To facilitate querying, Sesame supports the storage of large quantities of RDF and RDF Schema information. The RDF is parsed using the SiRPAC parser, and stored in the Object-Relational DBMS PostgreSQL. A public demo server running Sesame is available for experimentation.
URL: http://sesame.aidministrator.nl/
References: - Jeen Broekstra, Arjohn Kampman. "*Query Language Definition*". On-To-Knowledge project deliverable 9. March 2001. <http://sesame.aidministrator.nl/doc/del9.pdf>
- Arjohn Kampman, Frank van Harmelen. "*Sesame's Interpretation of RDF Schema*". Aidministrator Nederland bv. Version 1.2. April 24, 2001.<http://sesame.aidministrator.nl/doc/rdf-interpretation.html>
- Jeen Broekstra, Arjohn Kampman, Frank van Harmelen. "*Sesame: a Generic Architecture for Storing and Querying RDF and RDF Schema*". To appear in the 1st International Semantic Web Conference (ISWC2002), June 9-12, 2002. Sardinia, Italy.
Documentation: http://sesame.aidministrator.nl/docs.jsp
Tutorial: http://sesame.aidministrator.nl/publications/rql-babysteps.pdf
Version: 3- Alpha
Platform: any platform supporting a Java 2 runtime environment
Demonstration: available at http://sesame.aidministrator.nl/demo.jsp
Pricing Policy: Open source tool under the terms of GNU Lesser General Public License (LGPL).

3.2.3 Inkling

The Inkling query engine was developed under partial funding from the Harmony[5] and Imesh[6] projects at the ILRT (Institute for Learning and Research Technology, University of Bristol) and can be used to create, query and display RDF documents. It is a Java™ implementation of SquishQL [section 3.1.2] created to be API and database-independent for testing the usefulness of SquishQL for comparatively small-scale projects. To ensure the validity of the input RDF data, it uses and upgrades the SiRPAC parser. Inkling can be used with almost any RDF database implementation written in Java (either in-memory or using some persistent storage) and uses the JDBC interfaces to make SquishQL queries.
URL: http://swordfish.rdfweb.org/rdfquery/
References: -Miller,L. "*Inkling: RDF Query using SquishQL*". <http://swordfish.rdfweb.org/rdfquery>
Documentation: http://swordfish.rdfweb.org/rdfquery/documents.html

[5] http://metadata.net/harmony/index.html
[6] http://www.imesh.org/

Tutorial: not available
Version: alpha release (0.50)
Platform: any platform supporting Java
Demonstration: available at http://swordfish.rdfweb.org/rdfquery/demos.html
Pricing Policy: freely released under the GPL or a modified version of the MPL Licenses.

3.2.4 rdfDB

Implemented by R.V. Guha, rdfDB is intended to be a simple, scalable, open-source database for RDF metadata. Although at an early stage, the goals of the rdfDB implementation are to support a graph-oriented API via a textual query language a la SQL and to provide support for RDF ontologies and some basic forms of inference. The rdfDB uses a high-level, simple graph matching, triple-based SQL-like query language. This query language differs slightly in syntax from SquishQL [section 3.1.2] and also does not contain the constraints on the variables used by SquishQL. rdfDB is designed to act as a cache for RDF, RSS, edge-labelled XML and other data out on the network. To facilitate this, it supports the ability to load the contents of a URL into the database.
URL: http://guha.com/rdfdb/
References: Edd Dumbill. "*Putting RDF to Work*". Article on XML.com. August 09, 2000. (http://www.xml.com/pub/a/2000/08/09/rdfdb/)
Documentation: http://guha.com/rdfdb/
Tutorial: not available
Version: 0.46
Platform: Unix (linux, bsd, solaris)
Demonstration: not available
Pricing Policy: sources are available under the Mozilla Public License.

3.2.5 RDFStore

RDFStore is a set of Perl modules to manage RDF model databases in an easy and straightforward way. It consists of a Perl API, a streaming SiRPAC parser and a generic hashed data storage custom-designed for the RDF model. The storage subsystem allows transparent storage and retrieval of RDF nodes, arcs and labels from a variety of storage systems, i.e., either from an in-memory structure, from the local disk or from a very fast and scalable remote storage. Currently, it supports several different persistent storage models such as SDBM and BerkeleyDB. RDFStore implements the SquishQL language [section 3.1.2] to query RDF repositories and all query-filtering operations on the values are processed using pure Perl regular expressions. The query parsing, processing and execution is performed on the client side, which makes the DBMS server back-end much more generic and lightweight. RDFStore contains a module to make basic RDF Schema inference on triples. By using free-text words present into literal values it is possible to select the nodes matching a query criteria in a much more selective way.
URL: http://rdfstore.sourceforge.net/
References: not available
Documentation: http://rdfstore.sourceforge.net/documentation/api.html
Tutorial: not available
Version: 0.42
Platform: tested on FreeBSD and Linux, can run on any platform running Perl
Demonstration: available at http://rdfstoredemo.jrc.it/
Pricing Policy: free distribution.

3.2.6 Extensible Open RDF (EOR)

The Extensible Open RDF constitutes an open source project resulting from the cooperation of the OCLC Office of Research and the Dublin Core Metadata Initiative. Its goal is to facilitate the rapid development of RDF applications by providing generic interface design capabilities with focus on the discovery, management, integration and navigation of metadata. It consists of a collection of extensible Java classes and services, which serve as a code base, demonstrating by example functions and services common to RDF applications, i.e., metadata capture, search engines, etc. Thus, the base level functionality supported by this toolkit includes the creation, deletion and management of RDF databases. The current release provides services designed to validate RDF, to infuse RDF instance data into RDF databases, to build and search via triple-matching with wildcards RDF triple stores and render RDF data using XSLT. Hence, it provides the basic building blocks for supporting search

services, topic-maps, site-maps, annotation environments and semantic metadata registries based on RDF.
URL: http://eor.dublincore.org/
References: not available
Documentation: http://eor.dublincore.org/project_docs.html
Tutorial: not available
Version: 1.01
Platform: all platforms running Java
Demonstration: available at http://wip.dublincore.org:8080/eor/index.html
Pricing Policy: software available under the Dublin Core Open Source Software License.

3.2.7 Redland

Developed at the University of Bristol, Redland is a library that provides a high-level interface for storing, querying and manipulating RDF models. Redland implements each of the RDF model concepts in its own class, thus providing an object-based API for them. Some of the classes providing the parsers, storage mechanisms and other elements are built as modules that can be added or removed as required. In a nutshell, Redland provides a modular, object based library written in C, Perl, Python and Tcl, Java interfaces for manipulating the RDF Model and parts (Statements, Resources and Literals), parsers for reading RDF/XML and other syntaxes (DAML+OIL and N-Triples[7]), storage mechanisms for models in memory and on disk via Sleepcat/Berkeley DB, query APIs for the model by Statement (triples) or by Nodes and Arcs and streams for construction, parsing and de/serialisation of models.
URL: http://www.redland.opensource.ac.uk/
References: Dave Beckett. "*Design and Implementation of the Redland RDF Application Framework*". WWW10 Presentation. 2001-05-03.
Documentation: http://www.redland.opensource.ac.uk/docs/
Tutorial: not available
Version: 0.9.10
Platform: Linux, Solaris, OSF/1 Alpha, FreeBSD, MacOS X
Demonstration: available at http://www.redland.opensource.ac.uk/demo/
Pricing Policy: free software/open source software released under the GNU Lesser General Public License (LGPL) Version 2 or the Mozilla Public License V1.1.

3.2.8 Jena

Developed by the Hewlett-Packard Company, Jena is a collection of RDF tools written in Java that includes: a Java model/graph API, an RDF Parser (supporting an N-Triples filter), a query system based on RDQL [section 3.1.2], support classes for DAML+OIL ontologies and persistent/in-memory storage on BerkeleyDB or various other storage implementations. Due to its storage abstraction, Jena enables new storage subsystems to be integrated. To facilitate querying, Jena provides statement-centric methods for manipulating an RDF model as a set of RDF triples and resource-centric methods for manipulating an RDF model as a set of resources with properties, as well as built-in support for RDF containers. The current toolkit does not provide any inferencing mechanisms, since the query language used, i.e., RDQL, does not provide inference.
URL: http://www.hpl.hp.com/semweb/jena-top.html
References: B. McBride. "*Jena: Implementing the RDF Model and Syntax Specification*". In: Steffen Staab et al (eds.): Proceedings of the Second International Workshop on the Semantic Web-SemWeb2001. May 2001.
Documentation: http://www.hpl.hp.com/semweb/javadoc/index.html
Tutorial: not available
Version: 1.3.2
Platform: any platform with Java 1.2 and up runtime environment. Tested on MS Windows and Linux
Demonstration: not available
Pricing Policy: source code distribution under (a BSD style) Jena License.

3.2.9 RDF Gateway

The Resource Description Framework Gateway (RDF Gateway), developed by Intellidimension Company, is a distributed data semantic query service and inference infrastructure. It follows a

[7] http://www.w3.org/2001/sw/RDFCore/ntriples/

modularised approach and is comprised of two basic modules: Data Services, that translate structured data into a knowledge base of RDF triples, and a Query Service that takes queries and inference rules, processed through a query Language, namely RDFQL [section 3.1.3], and applies them to the knowledge base through a logic layer. In particular, an RDF Query Service (RDFQS) processes RDFQL queries from applications, while an RDF Data Service (RDFDS) eposes underlying data via an RDFDS Connection Interface. The Data Service module can interface with any data source, i.e., RDF structured data, XML files via http, databases, or email accounts, all of which can be regimented into RDF triples. This ability comes from the full exploitation of RDF's abstract data model, which offers a logical view into any structured datasource. All of the RDF statements generated by the Data Services constitute the common knowledge base, which cannot be considered a data repository, but a dynamic knowledge representation framework. The role of the Query Service, on the other hand, is to perform semantic queries on the knowledge base, and extend the knowledge base with the addition of inference rules. Since the Data Services can compile multiple data sources into a single knowledge base for querying and knowledge discovery, independently authored schemas can yield new information about common elements.
URL: http://www.intellidimension.com/RDFGateway/beta2/
References: not available
Documentation: http://www.intellidimension.com/RDFGateway/Docs/
Tutorial: http://www.intellidimension.com/RDFGateway/Docs/rdfqlgettingstarted.asp
Version: 0.6
Platform: Windows NT/2000
Demonstration: available at http://www.intellidimension.com/itdsw/default.asp
Pricing Policy: beta version of RDF Gateway available under License.

3.2.10 TRIPLE

The TRIPLE query engine constitutes the implementation of the TRIPLE query language [section 3.1.6]. Apart from the features of the query language, TRIPLE also contains a standalone DAML+OIL implementation with the following features: it parses DAML+OIL ontologies with Jena, provides output in various syntaxes (LISP, XML for FaCT DTD, while others can easily be added), supports an external DL classifier that can be automatically invoked (at the moment, it supports RACER but FaCT will follow) and the output from the DL classifier (taxonomy only) can be returned in various formats: DAML, LISP, XML, DOT (for visualization with GraphViz [14]). The set of the facilities presented make TRIPLE a powerful inference and querying engine.
URL: http://triple.semanticweb.org/
References: Michael Sintek, Stefan Decker. "*TRIPLE-An RDF Query, Inference, and Transformation Language*". In Proceedings of the Deductive Databases and Knowledge Management Workshop (DDLP' 2001). Japan, October 2001.
Documentation: not available
Tutorial: not available
Version: 2002/03/14
Platform: platforms supporting Java
Demonstration: available at http://ontoagents.stanford.edu:8080/triple/index.html
Pricing Policy: distributed under the terms of The Semantic Web Foundation for Open Source Software (SFO) License.

3.2.11 KAON Tool Suite

The Karlsruhe Ontology (KAON) tool suite has been developed in the context of the KAON Semantic Web infrastructure. To access an ontology, KAON tools use a programmable interface in Java called KAON-API. The same interface is implemented for several storage mechanisms, thus independence from the data store (database, text file, KAON server or another server such as RQL [section 3.1.1]) is achieved. This allows us to use the KAON front-ends (e.g., OntoMat - an authoring and annotation tool, KAON-CRAWL – a RDF crawler, etc) with different ontology storage tools. The main-memory based implementation of the KAON-API maps directly onto the RDF-API (an implementation of the graph-model for processing RDF, see http://www.w3.org/RDF/Validator/). Due to the existence of different representation languages for ontologies, the KAON-API tries to be (as far as possible) representation language neutral. The user can work with ontologies in different representation languages as long as the representation primitives have been defined to be semantically equivalent to the primitives of the KAON language that especially works with RDF Schema and DAML+OIL ontologies. The KAON vocabulary builds on an extension of the RDFS vocabulary (from RDF-API).

This backwards compatibility allows KAON ontologies to be treated as extensions of RDF and RDF Schema ontologies. However, KAON-API only supports some simple queries for browsing the ontology (e.g., get all classes or get all resources of a class) and does not allow specifying a filter or an expression path within the query (i.e., users will have many difficulties for exploring the ontology).
URL: http://kaon.semanticweb.org/
References: - Alexander Maedche, Boris Motik and Raphael Volz. "*KAON-A Framework for Semantics-based E-Services*". Institute AIFB, University of Karlsruhe.
- Siegfried Handschuh, Alexander Maedche, Ljiljana Stojanovic and Raphael Volz. "*KAON-The KArlsruhe ONtology and Semantic Web Infrastructure*". White paper available at http://kaon.aifb.uni-karlsruhe.de/white-paper
Documentation: http://kaon.aifb.uni-karlsruhe.de/documentation
Tutorial: not available
Version: 17/01/2002
Platform: any platform supporting Java
Demonstration: not available
Pricing Policy: software available under KAON license

3.2.12 Cerebra®

Cerebra® is an RDF Inference Engine (an automated reasoner) developed by Network Inference. Its supporting facilities are similar to the University of Manchester's FaCT Description Logics engine, with the difference that it supports instances and concrete datatypes. Cerebra allows for collaboration and has been reported to collaborate successfully with OilEd (version 2.0 and above) and Protegé-2000 with the OIL plug-in. Demonstration versions of Cerebra® for all major platforms (Windows and Linux) are available from the download section of the Network Inference website.
URL: http://www.networkinference.com/
References: not available
Documentation: http://www.networkinference.com/products.asp?id=54&menu=7
Tutorial: not available
Version: 1.2
Platform: Windows, Linux
Demonstration: not available
Pricing Policy: evaluation copy available under Cerebra Licence.

3.2.13 Empolis K42

Empolis K42 Knowledge Server, deployed by Empolis UK Ltd with Jini and RMI technology, constitutes a collaborative, web-based integrated authoring environment for capturing, expressing and delivering knowledge. It is based on the Topic Map standards technology and is able to import, export and merge Topic Maps (XTM) [20]. Furthermore, in order to empower the applicability of Topic Map technology to knowledge management problems, it supports a number of features, such as a framework for topic map visualization and navigation, a 100% JAVA Topic Map modeling API, a persistence model along with the Data Fetcher architecture, which allows k42 to be configured to connect to any data source apart from using its native storage mechanism, and a first Topic Map Query Language [section 3.1.8] implementation. In order to support the use of complex knowledge relationships in Topic Maps, k42 delivers also an inferencing engine interface that allows Topic Map developers to create advanced logic applications. Inferencing allows the creation of associations based on defined rules, which are defined in terms of topics and associations. When the rules are applied, the inference engine looks for combinations of topics that meet rules and associate them in the way that the rule defines. This means that not all of the associations in a Topic Map need to be explicitly created but can be inferred, thus facilitating the creation of new information from existing one.
URL: http://k42.empolis.co.uk/
References: not available
Documentation: http://k42.empolis.co.uk/
Tutorial: not available
Version: 1.1.1
Platform: all platforms supporting Java
Demonstration: available at http://k42.empolis.co.uk/demo/demo.html
Pricing Policy: evaluation copy of k42 available for free download under Empolis UK Ltd Licence.

3.2.14 Ontopia Knowledge Suite

The Ontopia Knowledge Suite, developed by the homonymous company, is a set of tools for building, maintaining and deploying topic map-based applications. It consists of three main products, namely the Ontopia Topic Map Engine, the Ontopia Navigator Framework and the RDBMS Backend Connector and of several smaller add-on components, while more products and components are under development. The Topic Map Engine, developed in Java SDK, constitutes the building block of the Suite, since it loads, stores, keeps track of the topic maps and provides interfaces through which applications can access and manage the topic maps. The engine has APIs for importing and exporting topic maps to and from XML documents. There is full support for the XTM 1.0 format and all its features, as well as for an XML version of the ISO 13250 format. There is also support for validating XTM topic maps against a DTD. To support the storage of topic maps using relational technology, the Suite provides the RDBMS Backend Connector, which enables the storage, access and modification of topic maps in relational databases (most RDBMS servers are supported). The query engine, which supports the *tolog* topic map query language [section 3.1.9], the schema tools, which implement a topic map schema language, and the full-text integration tool, which adds support for full-text search of topic maps are the add-on components that provide the Engine with additional power for advanced applications. The Suite also supports the development of topic map web applications with the Topic Map Navigator, a web application framework based on Java Servlets and Java Server Pages (JSP) offering a set of tag libraries and Java components for this purpose. A demonstration tool built with the Navigator Framework is the Omnigator, a topic map browser that can be used to browse any topic map. Omnigator is freely available by Ontopia, but it also forms part of the Ontopia Knowledge Suite.
URL: http://www.ontopia.net/solutions/products.html
References: "*The Ontopia Knowledge Suite: An introduction*". WHITE PAPER (Version 1.3), MARCH 2002. <http://www.ontopia.net/ontopia/texts/product-wp.html>
Documentation: not available
Tutorial: not available
Version: 1.3
Platform: platforms supporting Java 1.3
Demonstration: An Ontopia Topic Map Engine and Navigator demo is available at http://www.ontopia.net/solutions/demos.html
Pricing Policy: A free evaluation version of the Ontopia Omnigator is available at http://www.ontopia.net/download/index.html. The OKS is offered in three different editions, each of which is available under either a Developer License or a Runtime License.

4. Comparison of Query Languages and Storage Tools

The purpose of this section is to present a comparison of ontology query languages and storage/ querying tools against the evaluation frameworks adopted in Section 2. We have already given a brief description with basic descriptive criteria of selected ontology query languages and accompanying storage and querying tools based on three standards: RDF/S [17,4], DAML+OIL [9] and Topic Maps [16]. However, our interest as far as the evaluation of querying languages is concerned, will be in RDF/S querying languages, due to the fact that query languages for the other ontology standards are still ongoing. On the contrary, the evaluation framework designed for comparing tools will be applied on the set of the presented tools, since our orientation in this case has been mainly technical.

The context of storing and querying knowledge has significantly evolved due to the wide acceptance of the Web. The effort of RDF/S and other web-based metadata standards has been to satisfy the new requirements posed in this context by providing a common framework for the encoding and exchange of resource-related knowledge. In parallel with the research work dealing with issues related to syntax and semantics of RDF/S, efforts have been made to provide querying mechanisms for extracting information from description bases encoded in this standard. However, the RDF/S modeling primitives are substantially different from those defined in traditional database models such as object, relational or semi-structured, a fact that calls for different treatment. Many query languages, even those for XML (e.g., LOREL[1], StruQL[11], XML-QL[10], XML-GL[5], Quilt[13] or XQuery[6]), fail to capture the semantics of RDF description bases. This reason has lead many research contributions on creating new languages for querying RDF, namely RQL [3.1.1], SquishQL/RDQL [3.1.2], RDFQL [3.1.3], VERSA [3.1.5], RDFPath [3.1.4] and TRIPLE [3.1.6]. Apart from RDF/S query languages, two other types of

languages situated in the same context and also used for the comparison in this chapter are Description Logics[8] and Topic Map [16] query languages. Clearly, the above languages do not rely on the same data modeling and each language has different characteristics. To facilitate understanding, **Table 1** presents a general comparison for all these languages on some basic criteria describing the language characteristics.

Criteria / Query Lang.	Standard	Data model	Language of origin	Closure of queries	Orthogonality of input/ output data	Generality
RQL	RDF/S	Graph	OQL like	Yes	Yes	No
SquishQL/RDQL	RDF/S	Triple	SQL like	No	No	No
RDFQL	RDF/S	Triple	SQL like	No	No	No
RDFPath	RDF/S	Tree	XPath like	No	No	No
VERSA	RDF/S	Graph	LISP like	Yes	No	No
TRIPLE	RDF/S	Triple	F-logic like	No	No	No
Description Logics QLs	DAML/OIL	Triple	DL like	No	No	No
TMQL	Topic Maps	Graph	SQL like	No	No	Yes
Tolog	Topic Maps	Triple	Datalog like	No	No	No

Table 1: Ontology Query Languages

As we can note from **Table1**, most of the query languages rely on a triple data model, i.e., they use collections of statements of the form <subject, predicate, object>[9] to encode RDF/S description bases and Schemas. RQL, Versa and TMQL follow a graph data model, which enables them to perform more complex queries over the resource description graphs. RDFPath, on the other hand, adopts a tree model and its constructs enable to specify paths between two arbitrary nodes of a given RDF graph. Triple as well as tree data models necessitate exact knowledge of the RDF graph structure in order to perform querying, knowledge that is not need in most graph-based data models. Furthermore, languages like TRIPLE, Description Logics QLs and Tolog rely on logic-based models (i.e. we can define inference rules within these languages). A partial support of user-defined inference rules is also provided by RDFQL, e.g., for capturing traversable transitive properties. The rest of the languages follow either an SQL or an OQL approach, mainly with the purpose to provide constructs and functionality commonly known to database developers.

Regarding the respect to primary properties, RQL is the only language having both the properties of *closure* and *orthogonality*, i.e., supports functional composition of queries (i.e., results of a query could be used as input of another query). It also permits any kind of data as input and output of queries. VERSA also permits the composition of queries. However, the majority of studied languages, except RQL, do not satisfy the *orthogonality* property because they privilege some kind of data as input or output of data. For instance, the languages using the triple data model (e.g., SquishQL/RDQL, RDFQL, VERSA) take a set of triples as input and return as output a bag/list of resources or literal values. In contrast, there are not privileged input or output data for RQL, i.e., input or output data can be a class, a property, a resource or a container/literal value, etc. RQL however lacks the property of *generality*, i.e., the ability to support *all* the primitives of the ontology/metadata model. In particular, RQL exploits most of the RDF/S modeling constructs, e.g., single/multiple inheritance, single/multiple instantiation, container values and typed literal values but it does not support reification. In fact, Table 1 indicates that none of the RDF/S querying languages respects the *generality* property, as most of them either do not capture the reification mechanism (only TRIPLE supports it) or they do not support container values. On the contrary, TMQL, a language for ontologies in the Topic Maps standard, seems to support all the modeling primitives of its underlying standard (e.g., support of topics, topic types, associations, association roles).

[8] http://www.dl.kr.org/

[9] The structure of the statements may differ in each query language. Most of them follow the <subject, predicate, object> convention, while others use triples of the form <predicate, subject, object>.

Table 2 hosts a comparison of RDF/S query languages according to five axes: *Modeling constructs* supported, *Ontology Querying*, *Data Querying*, *Data/Ontology Querying* and *Additional Features* provided. Regarding the first comparison axis (***Modeling constructs***), we can observe from Table 2 that all query languages taking part in the comparison, i.e., RQL, SquishQL, RDFQL, RDFPath, VERSA and TRIPLE, support namespaces and multiple schemas, as well as multiple inheritance and instantiation. This means that all query languages can perform queries on resources described in terms of classes and properties from multiple namespaces and that the underlying data model employed in each case is able to capture multiple inheritance and instantiation. However, all query languages do not support container values and reification. In fact, reification is only supported by TRIPLE. Furthermore, most query languages support the basic data types, i.e., strings and integers, with the exception of RQL, VERSA and RDFQL that provide more elaborated data types, e.g., reals, dates, URIs or enumerated types.

We also compare the RDF/S querying languages in terms of their ability to perform Ontology ***Querying***. The ability to traverse the ancestor/descendant hierarchies of classes and properties requires the use of ontology (schema) knowledge and only RQL and TRIPLE can perform it transitively. Moreover, only RQL can pose sophisticated filtering conditions on class/property hierarchies. Apart from (in)equality and "like" conditions on the names of ontology constructs, RQL can also pose selection filters based on subsumption testing, i.e., on the subsumption relations among classes or properties, and on the namespaces in which these ontology constructs are defined. Furthermore, the ability of query languages to perform ***Data querying*** is of major importance. Based on this set of criteria, we can note from Table 2 that all query language provide constructs for finding the extent of a class or property, either directly or transitively. What most query languages do not support is set-based operations (union, intersection, difference) as well as arithmetic operations on data values. The former is mainly attributed to the fact that they do not support container values constructors (only RQL nad VERSA feature this facility). Furthermore, complete Boolean filters, i.e., conjunction, disjunction and negation, are only supported by RQL, RDFQL, VERSA and TRIPLE, since the other query languages support only conjunction.

The set of the criteria referring to ***Data/Ontology Querying*** are indicative of the expressive power of an RDF query language, i.e., its ability to combine data and ontology (schema) querying at the same time. The basic criterion for judging this ability is the use of generalized path expressions. Generalized path expressions are very useful primitives because they allow data and ontology to be uniformly queried. As indicated from Table 2, only RQL is capable of incorporating knowledge from ontologies into data querying. In fact, RQL features generalized path expressions with variables on labels of both nodes and edges. On the contrary, TRIPLE and RDFPath can support only data paths, while VERSA provides transitive traversal of properties. Furthermore, RQL and TRIPLE are the only query languages supporting existential or universal quantifiers. Moreover, the ability to perform nested queries is only supported by RQL and Versa.

To evaluate the effectiveness of the query languages when used in large-scale Semantic Web applications we can also use a set of criteria referring to ***Additional Features*** supported by the query languages. Table 2 records whether the query languages under evaluation support aggregate, grouping and sorting functions. More specifically, RDFQL supports only a count function, VERSA supports min and max, while RQL features min, max, count, average and sum functions, as known from relational databases. Sorting functions are only supported by RDFQL and VERSA, with both of them also supporting built-in data functions for e.g., converting date and string values or calculating arithmetic values. A functionality not supported by any of the query languages is the grouping of querying results. Furthermore, RDFQL, apart from providing data definition capabilities (e.g., creation of tables), provides also primitives for defining views and user-defined inference rules. User-defined inference rules are also definable in TRIPLE and can be used for instance, to inversely traverse properties. Lastly, among additional features we can include the ability of the user to define arbitrary functions. This capability is only viable in RDFQL.

Table 4 is used to evaluate the technical features of practical implementations supporting storage and querying facilities for description bases and schemas encoded in RDF/S [17,4], DAML+OIL [9] or Topic Maps [16]. This comparison gives an overview of the 14 ontology storage and querying tools studied in section 3.2. To facilitate comparison of technical features, **Table 3** provides a synopsis of the basic criteria used for the description of the tools. As one can see, most of the tools provide a web-based demonstration of the facilities they support, while open source distribution in the form of public licenses and the existence of documentation assist third-party developers when building applications on top of these tools. On the contrary, the lack of tutorials does not facilitate the overview of the system's.

	Languages / Criteria	RQL	SquishQL	RDFQL	RDFPath	VERSA	TRIPLE
Modeling Constructs	Namespaces/ Multiple schemas	YES	YES	YES	YES	YES	YES
	Data types	strings, integers, reals, dates, URI, enumerated and thesauri types	strings and integers	strings, integers, reals, dates and URI types	strings	strings, numbers, URI, lists, sets, booleans	strings, integers
	Multiple inheritance/ Instantiation	YES	YES	YES	YES	YES	YES
	Container values	YES	YES	YES	YES	NO(?)	NO
	Reification	NO	NO	NO	NO	NO	YES
Ontology Querying	Ancestor/ descendant traversal of class/property hierarchies	YES	NO (only direct)	NO (only direct)	NO (only direct)	NO (only direct)	YES
	Filtering conditions on class/property hierarchies	(in)equality, like, subsumption check, namespace querying	like (~), equality	lexicographical ordering on class /property names, equality	equality	(in)equality, string containment	(in)equality, subsumption check
Data Querying	Class/property extent queries	YES	YES (only direct)	YES (only direct)	YES (only direct)	YES	YES
	Complete Boolean Filters (negation, (con/dis)junction)	YES	NO (only conjunction)	YES	NO (only conjunction)	YES	YES
	Set-based operations	YES	NO	NO	NO	YES	YES
	Arithmetic operations	YES	NO	NO	NO	NO	NO
	Container values constructors	YES	NO	NO	NO	YES	NO
Data/ Ontology Querying	Generalized path expressions	YES	NO	NO	NO	NO	NO
	Existential/ Universal quantifiers	YES	NO	NO	NO	NO	YES
	Nested queries	YES	NO	NO	NO	YES	NO
Additional Features	Aggregate functions	YES	NO	YES (only count)	NO	YES	NO
	Grouping functions	NO	NO	NO	NO	NO	NO
	Sorting functions	NO	NO	YES	NO	YES	NO
	Built-in data functions	YES (thesauri terms)	NO	YES (math/ string/date)	NO	YES (conversion functions)	NO
	Arbitrary function support	NO	NO	YES	NO	NO	NO
	User-defined inference rules	NO	NO	YES	NO	NO	YES
	View definition primitives	NO	NO	YES	NO	NO	NO

Table 2: Expressive power of RDF/S query language

	Ref.	Doc.	Tutorial	Version	Platform	Demo	Pricing Policy
ICS-RDFSuite	YES	YES	YES	1.5	Solaris/ Linux	YES	GPL compatible License
SESAME	YES	YES	YES	3-Alpha	Any (Java)	YES	LGPL License
INKLING	YES	YES	NO	Alpha	Any (Java)	YES	GPL/MPL License
RDFDB	YES	YES	NO	0.46	Linux, Bsd, Solaris	NO	Mozilla License
RDFSTORE	NO	YES	NO	0.42	Any (Perl)	YES	Free distribution
EOR	NO	YES	NO	1.01	Any (Java)	YES	Dublin Core Open Source License
REDLAND	YES	YES	NO	0.9.10	Linux, Solaris, OSF/1, FreeBSD, MacOS X	YES	LGPL/Mozilla License
JENA	YES	YES	NO	1.3.2	Any (Java)	NO	Jena License
RDF GATEWAY	NO	YES	YES	0.6	Windows NT/2000	YES	RDF Gateway License
TRIPLE	YES	NO	NO	2002/03/14	Any (Java)	YES	Semantic Web Foundation for Open Source License
KAON	YES	YES	NO	2002/01/17	Any (Java)	NO	KAON License
CEREBRA	NO	YES	NO	1.1	Windows/ Linux	NO	Cerebra License
Empolis k42	NO	YES	NO	1.1.1	Any (Java)	YES	Empolis Ltd License
Ontopia KS	YES	NO	NO	1.3	Any (Java 1.3)	YES	Developer/ Runtime Licenses

Table 3: Overview of ontology storage/querying tools

Furthermore, most of the tools presented can be used with any software platform (supporting though the Java runtime environment). This can also be concluded from **Table 4**, when referring to the language used for the implementation of the tool. As we can see from column **Implementation Language**, 10 out of 14 tools were developed with Java, thus profiting from the platform independence endorsed by Java. The next most popular implementation language of ontology tool developers is C, a language whose stability and functionality has already been tested for years in the deployment of large-scale applications. The popularity of Java as an implementation medium can also be noted when examining the **API support** provided by the tools. However, apart from Java and C, the developers provide a set of APIs written in different languages, e.g., Perl, Python and Tcl in an attempt to facilitate the collaboration of their tool with other applications. These APIs provide functions for querying and updating description bases and can be used for interfacing with clients. Hence, they offer third party-developers the ability to deploy their applications on top of these tools. For this reason, the criterion of API support can be thought of as a measure of the extensibility and the degree of collaboration with other applications characterizing by the tool.

One general comment that can be made regards the tendency to provide storage and querying support for RDF/S description bases. Although a number of tools also provide support for DAML+OIL, the focus of developers has been to provide facilities for description bases encoded in the RDF/S standard, while the contributions for the Topic Map standard are also few. Most of the **query languages** rely on a triple-based model, such as SquishQL/RDQL, RDFQL or TRIPLE (their functionality has already been compared above). EOR and REDLAND do not implement a specific query language and they support a straiforward triple-matching mechanism for querying. Furthermore, most of the presented tools exploit relational technology to **store** RDF/Topic Map data. In fact, (O)RDBMSs are preferred for persistent data storage (namely PostgreSQL, BerkeleyDB, MySQL), since there is no specific database implementation for RDF or Topic Maps native model (with the exception of the Empolis K42, which implements its own storage system). The BerkeleyDBN and PostgresSQL are mostly preferred due to their free distribution. Apart from persistent storage, the majority of the tools also support in-memory storage, in an attempt to minimize the query response time. The **scalability** and **performance** features obtained from a survey of the W3C[10] are quite preliminary and more exhaustive comparative performance tests are required in order to draw useful conclusions. In addition, we consider the ability of the tool to support **updates**, both in the ontology and in the (meta)data. **Table 4** indicates that the majority of the tools are able to perform updates more by inserting (or deleting) new information to the database and less by modifying existing one.

[10] http://www.w3.org/2001/05/rdf-ds/DataStore

Tools	Query Language	Impl. Language	Storage DB	Inference support	Update support (Ontology + Data)	API support (Querying + Updating)	Scalability / Performance	Export data format
ICS-RDFSuite	RQL	Java/C++	ORDBMS (SQL3 compliant, e.g., PostgreSQL)	Yes	Yes	C++/Java/SQL functions	DBMS scales linearly with the number of triples/ 505650 schema + 5331603 data triples= 5837253 triples total	RDF
SESAME	RQL*	Java	ORDBMS (PostgreSQL)	Yes	Yes	HTTP/SOAP	?	RDF
INKLING	SquishQL	Java	In-memory/persistence (supporting JDBC, e.g., SQL, Postgres SQL)	No	No	Java	?	Triples in ASCII
RDFDB	SquishQL*	C	Persistence (SleepyCat)	Yes	Yes	C, Perl	~20 million triples tested	Triples in ASCII
RDFSTORE	SquishQL	C, Perl	In-memory / persistence (e.g., file, BerkeleyDB, SDBM)	Yes	No	Perl	1470000 triples stored in a ~98MB database/ ~183 read operations/second	N-Triples, RDF
EOR	Triple-matching	Java	Persistence (SQL databases, e.g. MySQL)	No	Yes	HTTP, Java, SQL/JDBC	?	Triples rendered with XSL
REDLAND	Triple-matching	C	In memory / persistence (SleepyCat / BerkerleyDB)	No	Yes	Java, C, Perl, Python, Tcl	tested with 1.5M stored statements/ query speed is 6,200 statements/ second	Triples
JENA	RDQL	Java	In-memory / persistence (e.g., BerkeleyDB, Interbase, PostgreSQL)	No	Yes	Java	In-memory storage has been used with 600K statements/ for the SQL store is around 10ms/statement load, 1-7ms/ returned-statement search	Triples in ASCII
RDF GATEWAY	RDFQL	?	RDBMS	Yes	Yes	Microsoft ADO, JDBC	?	Triples in ASCII
TRIPLE	TRIPLE	Java	In-memory	Yes	?	Java	?	Lisp, XML for FaCT DTD, DOT, DAML, ASCII
KAON	F-logic	Java, Python	In-memory / persistence (KAON server/ Files, RDBMS)	Yes	Yes	Java	?	?
CEREBRA	DL-based	Java	Distributed data (CORBA)	Yes	-	Java	?	?
Empolis k42	TMQL	Java	Persistence storage (K42 Generic Store, other DBMS)	Yes	Yes	Java/RMI	~500MB tested/ 0.08 sec for look-up of an object by name for first access	Topic Maps (XTM)
Ontopia KS	Tolog	Java	In-memory / RDBMS / OODB	Yes	Yes	Java/J2EE	?	XTM, XML version of ISO 13250

Table 4: Technical table of ontology storage/querying tools

* Differs slightly from the original language

As far as **inference support** is concerned, we note that the majority of the studied tools provide some inference support especially for recursively traversing ontology class/property hierarchies as well as of data paths involving transitive properties. We can distinguish between two kinds of inference support: (a) seamless integration with the underlying query language engine (e.g., RQL and TRIPLE) or (b) loose coupling of the tool with an external inference engine. For instance, RDFSTORE and RDFGATEWAY enrich their triple-matching mechanism with an inference engine. Finally, the most common export data format is triples in ASCII. Some tools, e.g., the ICS-RDFSuite and Sesame, are able to **export** ontologies and query results in RDF, while Topic Maps tools export XTM topic maps. TRIPLE, on the other hand, due to its ability to support a variety of ontology/metadata standards outputs in Lisp, ASCII etc, while it can visualize query results (DOT files are used by GraphViz[14])

5. Conclusions and main recommendations

In the previous sections, we have briefly presented and compared a set of query languages and storage/ querying tools for specific knowledge representation formalisms, namely RDF/S [17,4], DAML+OIL [9] and Topic Maps [16], in terms of the evaluation frameworks presented in section 2. Due to the preliminary functionality status of query languages for DAML+OIL and Topic Maps, our focus has been on RDF/S query languages, which despite the fact that they commit to the same ontology/metadata standard, are based on quite different data representation paradigms (triples vs. trees vs. graphs) and they target quite different query functionality. One of the conclusions drawn from this survey is that the majority of the query languages do not yet support the complete set of modeling constructs offered by the above standards. Furthermore, the frontiers between querying and inferring capabilities offered by these languages are not clear. In most cases, inference is limited to a recursive traversal of ontology class/property hierarchies as well as of data paths involving transitive properties. We believe that the target functionality of the proposed query languages still has to be justified with respect to real large-scale Semantic Web applications.

In addition, from an industrial perspective, the various storage and query tools yet seem immature. In most cases, they are (academic) prototypes implementing parts of the query language they aim to support, while they do not provide the necessary programming/administration facilities in order to make them really operational into a working environment. Moreover, exhaustive scalability and performance figures are not yet available (with the exception of RDFSuite[2]).

In this context, we believe that an extensive set of use cases from large-scale Semantic applications is required, as conducted by the W3C XML Query Group [7]. The use of queries on data from different application contexts would not only give credible arguments about the target query functionality, but it could also provide a common testbed for the various implementations of query languages and storage tools. A first effort towards this direction has been proposed in [22].

6. References

[1] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener. "*The Lorel Query Language for Semistructured Data*". International Journal on Digital Libraries, 1(1): 68-88. April 1997.

[2] S. Alexaki, V. Christophides, G. Karvounarakis, D. Plexousakis. "*On Storing Voluminous RDF Descriptions: The case of Web Portal Catalogs*". In Proceedings of the 4th International Workshop on the Web and Databases (WebDB'01) - In conjunction with ACM SIGMOD/PODS, Santa Barbara, CA. May 24-25, 2001.

[3] Tim Berners-Lee, James Hendler, Ora Lassila. "*The Semantic Web*". Scientific American. May 2001. <http://www.sciam.com/2001/0501issue/0501berners-lee.html>

[4] D. Brickley, R.V. Guha. "*Resource Description Framework Schema (RDF/S) Specification 1.0*". W3C Recommendation. March 27, 2000. <http://www.w3.org/TR/rdf-schema>

[5] S. Ceri, S. Comai, E. Damiani, P. Fraternali, S. Paraboschi, and L. Tanca. "*XML-GL: a Graphical Language for Querying and Restructuring XML Documents*". In Proceedings of International World Wide Web Conference, Toronto, Canada. 1999

[6] D. Chamberlin, D. Florescu, J. Robie, J. Simeon, and M. Stefanescu. "*XQuery: A Query Language for XML*". Working draft, World Wide Web Consortium. June 2001. <http://www.w3.org/TR/xquery/>
[7] Don Chamberlin, Peter Fankhauser, Massimo Marchiori, Jonathan Robie. "*XML Query Use Cases*". W3C Working Draft, 20 December 2001. <http://www.w3.org/TR/xmlquery-use-cases>
[8] James Clark, Steve DeRose. "*XML Path Language (Xpath)*". W3C Recommendation, Version 1.0. 16 November 1999. <http://www.w3.org/TR/xpath>
[9] DAML+ OIL (March 2001) <http://www.daml.org/2001/03/daml+oil-index>
[10] A. Deutsch, M.F. Fernandez, D. Florescu, A. Levy, and D. Suciu. "*A Query Language for XML*". In Proceedings of the 8th International World Wide Web Conference, Toronto. 1999.
[11] M.F. Fernandez, D. Florescu, J. Kang, A.Y. Levy, and D. Suciu. "*System Demonstration - Strudel: A Web-site Management System*". In Proceedings of ACM SIGMOD Conf. on Management of Data, Tucson, AZ. May 1997. Exhibition Program.
[12] Ian Horrocks and Sergio Tessaris. "*Querying the Semantic Web: a Formal Approach*". To appear in the 1st International Semantic Web Conference (ISWC2002), June 9-12, 2002. Sardinia, Italy.
[13] D. Florescu, D. Chamberlin, J. Robie. "*Quilt: An XML query language for heterogeneous data sources*". In WebDB'2000, pages 53-62, Dallas, US. May 2000.
[14] GraphViz, open source graph drawing software: http://www.research.att.com/sw/tools/graphviz/
[15] Jeff Heflin, Raphael Volz, Jonathan Dale. "*Requirements for a Web Ontology Language*". W3C Working Draft. 7 March 2002.
[16] ISO/IEC 13250, Topic Maps <http://www.y12.doe.gov/sgml/sc34/document/0129.pdf>
[17] O. Lassila, R. Swick. "*Resource Description Framework (RDF) Model and Syntax Specification*". W3C Candidate Recommendation. February 1999. <http://www.w3.org/TR/REC-rdf-syntax>
[18] Ontology Inference Layer (OIL) <http://www.ontoknowledge.org/oil/>
[19] J.R. Ossenbruggen, H.L Hardman, L. Rutledge. "*Hypermedia and the Semantic Web: A Research agenda*". INS-R0105. May 31,2001
[20] Steve Pepper, Graham Moore. "*XML Topic Maps (XTM) 1.0"*. TopicMaps.org Specification. 2001.
[21] RDF Data Query Language <http://www.hpl.hp.com/semweb/rdql.html/>
[22] Jonathan Robie. "*The Syntactic Web- Syntax and Semantics on the Web*". XML Conference and Exposition 2001, December 9-14 2001, Orlando, Florida, USA.
[23] XML: Extensible Markup Language <http://www.w3.org/XML/>