A Foundation for a Mathematical Web Services Query Language: A Survey on Relevant Query Languages and Tools

Rebhi Baraka
Research Institute for Symbolic Computation (RISC-Linz)
Johannes Kepler University, Linz, Austria
rbaraka@risc.uni-linz.ac.at

September 2004

Abstract

In this report, we briefly present a survey on a number of query languages and tools that can serve as a basis for querying mathematical service descriptions. Our ultimate goal is to design and implement a mathematical services query language based on some selected features of these query languages.

Contents

1 Introduction

With the increased use of the Extensible Markup Language (XML) [8] as a central technology for storing, retrieving, and processing information, research and standardization communities began to develop and specify query languages for facilitating the search of XML-based documents deployed on the Web. The mathematical community began using XML for storing mathematical documents and bringing mathematics into the Web, e.g., as Web services, and started to develop query languages and tools for searching such documents. In the MathBroker project [25], we have developed the Mathematical Services Description Language (MSDL) [10] to describe mathematical Web services so that these services can be discovered by clients. To facilitate the process of publishing and discovering such services, we developed a registry [5] that can be used to publish service descriptions in MSDL so clients can discover them by means of browsing the registry or by querying it. Querying capabilities of the registry are still limited in the sense that they only treat metadata about the service descriptions but not the contents of the descriptions themselves. We need a query language that is able to query MSDL descriptions.

In this survey, we study some of the existing XML-based query languages and investigate their suitability as a basis for querying mathematical service descriptions. These languages range from simple node finding and path expressions to languages for RDF [33] and OWL [32] to even more specialized languages devised for mathematical documents. We present the query capabilities of the ebXML registry [17] because our query language implementation will be an extension to its query capabilities since our registry is an extension of the ebXML registry. We present an overview, features and query expressiveness of these languages. We do not present any performance comparison, as this needs a separate experimental study.

This study serves our goal to decide on the design and implementation of a query language for MSDL [10].

2 Querying in XML

First we address querying in XML documents.

2.1 Querying in the ebXML Registry

The ebXML registry currently supports two kinds of query capabilities: *Filter Query* and *SQL Query*. A query capability allows a client (an object of type `QueryManagerClient`) to search for or query different kinds of registry objects in the registry.

The following scenario takes place when performing a query:

- A client submits a query to the `QueryManager` of the registry by sending an `AdhocQueryRequest`.
- The `AdhocQueryRequest` contains a sub-element that defines a query in one of the supported registry query capabilities.
- The `QueryManager` sends an `AdhocQueryResponse` back to the client.
- The `AdhocQueryResponse` returns a collection of objects whose element type depends upon the `responseOption` attribute of the `AdhocQueryRequest`. These may be objects representing leaf classes in the registry information model [16], references to objects in the registry and/or intermediate classes in registry information model such as `RegistryObject` and `RegistryEntry`.

- Any errors in the query request messages are indicated in the corresponding AdhocQuery-Response message.

The AdhocQueryRequest syntax (as defined in the ebRIM Schema [15]) is as follows:

```
<element name="AdhocQueryRequest">
 <complexType>
  <complexContent>
   <extension base="rs:RegistryRequestType">
    <sequence>
     <element ref="query:ResponseOption"/>
     <choice>
      <element ref="query:FilterQuery"/>
      <element ref="query:SQLQuery"/>
     </choice>
    </sequence>
    <attribute name="federated"
               type="http://www.w3.org/2001/XMLSchemaboolean"
               default="false" />
    <attribute name="federation"
               type="http://www.w3.org/2001/XMLSchemaanyURI" />
    <attribute name="maxResults"
               type="http://www.w3.org/2001/XMLSchemainteger"
               default="- 1" />
    <attribute name="startIndex"
               type="http://www.w3.org/2001/XMLSchemainteger"
               default="0" />
   </extension>
  </complexContent>
 </complexType>
</element>
```

The FilterQuery and SQLQuery parameters specify a registry "Filter Query" and a registry "SQL Query" respectively. The optional maxResults parameter specifies a limit on the maximum number of results the client wishes the query to return. The required ResponseOption parameter allows the client to control the format and content of the AdhocQueryResponse to this request.

The AdhocQueryResponse syntax is as follows:

```
<element name="AdhocQueryResponse">
 <complexType>
  <complexContent>
   <extension base="rs:RegistryResponseType">
     <choice>
      <element ref="query:FilterQueryResult"/>
      <element ref="query:SQLQueryResult"/>
     </choice>
     <attribute name="startIndex"
                type="http://www.w3.org/2001/XMLSchemainteger" default="0" />
```

```
        <attribute name="totalResultCount"
                   type="http://www.w3.org/2001/XMLSchemainteger" />
       </extension>
      </complexContent>
     </complexType>
    </element>
```

The FilterQueryResult and SQLQueryResult parameters specify the result of a registry "Filter Query" and a registry "SQL Query" respectively. The optional startIndex integer value is used to indicate the index for the first result in the result set returned by the query, within the complete result set matching the query. The optional totalResultCount parameter specifies the size of the complete result set matching the query within the registry.

A client specifies a responseOption structure within an AdhocQueryRequest to indicate the format of the results within the corresponding AdhocQueryResponse. The responseOption syntax is as follows:

```
<complexType name="ResponseOptionType">
 <complexContent>
  <restriction base="http://www.w3.org/2001/XMLSchemaanyType">
   <attribute name="returnComposedObjects"
              type="http://www.w3.org/2001/XMLSchemaboolean"
              default="false" />
   <attribute name="returnType" default="RegistryObject">
    <simpleType>
     <restriction base="http://www.w3.org/2001/XMLSchemaNCName">
      <enumeration value="ObjectRef"/>
      <enumeration value="RegistryObject"/>
      <enumeration value="RegistryEntry"/>
      <enumeration value="LeafClass"/>
      <enumeration value="LeafClassWithRepositoryItem"/>
     </restriction>
    </simpleType>
   </attribute>
  </restriction>
 </complexContent>
</complexType>
```

The optional returnComposedObjects parameter specifies whether the registry objects returned should include composed objects. The optional returnType parameter specifies the type of RegistryObject to return within the response. Values for returnType are ObjectRef, LeafClass, RegistryObject, RegistryEntry, and LeafClassWithRepositoryItem. Next we describe each of the two query capabilities.

2.1.1 Filter Query Capability

A "Filter Query" has an XML syntax that provides simple query capabilities for the registry. It aims at reducing the amount of processing on the server side. Each query alternative is directed against a single class defined by the ebXML registry information model. The registry uses two types of filter queries depending on which classes are queried on:

- Queries on `RegistryObject` and `RegistryEntry` allow for generic queries that might return different subclasses of the class that is queried on. The result of such a query is a set of XML elements that correspond to instances of any class that satisfies the `responseOption` of the `AdhocQueryRequest`. For example `RegistryObjectQuery` with `responseOption LeafClass` will return all attributes of all instances that satisfy the query. This implies that response might return XML elements that correspond to classes like `ClassificationScheme`, `RegistryPackage`, `Organization` and `Service`.
- Queries on selected registry information model classes in order to define the exact traversals of these classes. Responses to these queries are accordingly constrained.

Each "Filter Query" alternative is associated with a registry information model binding that identifies a hierarchy of classes derived from a single class and its associations with other classes as defined by registry information model.

The ebXML registry specification [17] defines registry information model bindings that identify the virtual hierarchy for each "Filter Query" alternative. The semantic rules for each query alternative specify the effect of that binding on query semantics.

As an example, take the following `ExtrinsicObject` query to identify a set of extrinsic object instances as the result of a query over selected registry metadata. The syntax of such a query is as follows:

```
<complexType name="ExtrinsicObjectQueryType">
 <complexContent>
  <extension base="tns:RegistryEntryQueryType">
   <sequence>
    <element ref="tns:ExtrinsicObjectFilter" minOccurs="0" maxOccurs="1" />
   </sequence>
  </extension>
 </complexContent>
</complexType>
```

The `ExtrinsicObjectQueryResult` syntax is as follows:

```
<element name="ExtrinsicObjectQueryResult">
 <complexType>
  <choice minOccurs="0" maxOccurs="unbounded">
   <element ref="rim:ObjectRef" />
   <element ref="rim:RegistryEntry" />
   <element ref="rim:RegistryObject" />
   <element ref="rim:ExtrinsicObject" />
  </choice>
 </complexType>
</element>
```

The semantic rules (as stated in [17]) that define the results of the query are:

1. Let EO denote the set of all persistent `ExtrinsicObject` instances in the Registry. The following steps will eliminate instances in EO that do not satisfy the conditions of the specified filters.

(a) If EO is empty then continue to the next numbered rule.

(b) If an `ExtrinsicObjectFilter` is not specified then go to the next step, otherwise let x be an extrinsic object in EO. If x does not satisfy the `ExtrinsicObjectFilter` then remove x from EO. If EO is empty then continue to the next numbered rule.

(c) Let EO be the set of remaining `ExtrinsicObject` instances. Evaluate inherited `RegistryEntryQuery` over EO.

2. If EO is empty, then raise the warning: extrinsic object query result is empty; otherwise set EO to be the result of the `ExtrinsicObjectQuery`.

3. Return the result and any accumulated warnings or exceptions (in the `RegistryErrorList`) within the `RegistryResponse`.

2.1.2 SQL Query Capability

The optional `SQLQuery` element in the `AdhocQueryRequest` allows a client to submit complex SQL queries using the syntax for the `SQLQuery` of the registry as defined by subset of the SELECT statement as defined by [22] and [41]. The syntax of the registry query language is defined by a BNF grammar [17].

SQL queries are defined based upon a fixed relational schema [17] which is an algorithmic binding to the registry information model.

The result of an SQL query resolves to a collection of objects within the registry. Depending upon the `responseOption` parameter specified by the client on the `AdHocQueryRequest`, the result set is returned as an `ObjectRef`, `RegistryObject`, `RegistryEntry` or leaf registry information model.

The simplest form of an SQL query is based upon metadata attributes specified for a single class within the registry information model. For example, to retrieve the collection of `ExtrinsicObjects` whose name contains the word Acme and that have a version greater than 1.3, the following query is submitted:

```
SELECT eo.id from ExtrinsicObject eo, Name nm where nm.value LIKE '%Acme%' AND
       eo.id = nm.parent AND
       eo.majorVersion >= 1 AND
       (eo.majorVersion >= 2 OR eo.minorVersion > 3);
```

The query syntax allows for conjugation of simpler predicates into more complex queries as shown in the simple example above.

The SQL query schema defines a special views called `RegistryObject` and `RegistryEntry` that allow doing a polymorphic query against all `RegistryObject/RegistryEntry` instances.

Classification queries include `ClassificationNode`, `ClassificationScheme`, objects classified by a `ClassificationNode`, and classifications that classify an object. For example the following query retrieves the collection of extrinsic objects classified by specified `ClassificationNodes`:

```
SELECT id FROM ExtrinsicObject
WHERE
  id IN (SELECT classifiedObject FROM Classification
         WHERE
           classificationNode IN (SELECT id FROM ClassificationNode
```

```
                 WHERE path = '/Geography/Asia/Japan' ))
AND
 id IN
  (SELECT classifiedObject FROM Classification
   WHERE
     classificationNode IN (SELECT id FROM ClassificationNode
         WHERE path = '/Industry/Automotive'))
```

It retrieves the collection of extrinsic objects that are classified by the Automotive Industry and the Japan Geography. It will also contain any objects that are classified by descendents of the specified classification nodes.

Association-related queries include queries such as retrieving all associations with specified object as source, retrieving all associations with specified object as target, retrieving associated objects based on association attributes, and complex association queries. For example the following query selects associations that have a specific source object, target object and association type:

```
SELECT id FROM Association WHERE
       sourceObject = <id1> AND
       targetObject = <id2> AND
       associationType = <associationType>;
```

2.2 Querying in Xindice

Apache Xindice [44] is a database designed to store XML data. Xindice's main contribution is that, when searching through a large collection of documents, it can use indexes to intelligently select a set of candidate documents. Queries in Xindice provide functionality for looking up and retrieving documents or parts of documents using a variety of query languages. Updates of the database itself are also possible using queries. It has two native query languages, XPath [45] and XUpdate [47].

2.2.1 The Query Interfaces

When Xindice first initializes itself, it creates a query engine. The purpose of this query engine is to setup one or more query resolvers (query styles) to handle querying related to one particular query language, e.g. XPath or XUpdate. The interface `QueryResolver` has two implementations within Xindice, one for XPath and one for XUpdate, users can add their own implementations to support other query languages. The `QueryResolver` interface provides two important methods: `query()` and `compileQuery()`. Both model a complete invocation of a query on a collection. The difference between both methods is that `query()` invokes the query immediately and returns results, whereas `compileQuery()` encapsulates the invocation into a query object which can be used several times later on to invoke the query.

2.2.2 XPath Queries

The `XPathQueryResolver` class implements a query resolver for the XPath language. It provides two query methods: one for immediate execution and another for storing invocations for later use. Internally, `XPathQueryResolver` always compiles queries into an XPath query. Analyzing and compiling the query is actually handled by Xalan [43], not Xindice. Xalan (a processor for transforming XML documents into HTML, text, or other XML document types) contains XPath manipulation and evaluation classes, and Xindice uses them. When an XPath query is evaluated, a set of candidate documents is selected from the collection, then XPath is evaluated using the Xalan classes against each of these documents in turn, the document is loaded into a DOM [14] tree, the XPath query is evaluated against this DOM tree, and the results of all evaluations are aggregated and returned. When performing a query, it is possible to specify which documents should be considered. If this is done, then Xindice will use the provided set of documents as the candidate set, and execute the XPath query against each of them, reading them into memory first. If no explicit set of documents is specified, Xindice will try to locate an appropriate index based on XPath query. This index can then provide an intelligent set of candidate documents, and XPath is evaluated against only these documents. If no appropriate index is found, Xindice resorts to "brute force"; it evaluates the XPath expression against every document in the collection, thus effectively reading in, parsing and searching each document.

The following XPath query is run against all documents in the pebbles collection with "rock" node that has the type attribute = "hard".

```
xindice xpath -c /db/pebbles -q "/rock[@type='hard']"
```

The following XPath query is run against all documents in the pebbles collection with "rock" node that has the type attribute "hard" when the rock element is in the namespace `http://www.bedrock.com`.

```
xindice xpath -c /db/pebbles -s "br=http://www.bedrock.com" -q
"/br:rock[@type='hard']"
```

2.2.3 XUpdate Queries

XUpdate [47] queries are used to send update instructions to Xindice. An XUpdate query is actually a complete XML document which can contain any number of update instructions. The namespace bindings for prefixes used in the XUpdate instructions can be specified in the XUpdate string itself. The `XUpdateQueryResolver` class provides a query resolver for XUpdate queries just as the XPath resolver. When an XUpdate query is submitted with a specified set of documents, then all the XUpdate modification instructions are performed on each of the documents. If no documents are specified, Xindice proceeds as follows:

- For each of the modification instructions, it looks up the selector expression, that is an XPath expression that each XUpdate modification instruction contains to indicate which node in a document should be modified.
- Next, a complete XPath query is performed on the collection, using the `XPathQueryResolver` to find all nodes matching the XPath in documents in the collection, and these nodes containing documents are retained. Note that this step might make use of any indexes set up for XPath, as determined by the behavior of the XPath query resolver.
- The XUpdate modification instruction is performed on the retained set of documents.

To perform the updates, the set of documents is always processed as follows:

- Each document in the set is read in to a DOM tree in memory using the DOM compressor and BTree filer classes.
- This is updated in-memory DOM representation according to the modification instructions.

- Xindice writes the entire document back using the same DOM compressor and `B-Tree` filer classes.

The following sets the Amazon.com US dollar price of the book with an isbn of '123' to 40.00.

```
<xu:modifications version="1.0"
                  xmlns:xu="http://www.xmldb.org/xupdate">
    <xu:update select="/book[@isbn=?123?]/price[@currency=?US??
                   and @retailer=?Amazon?]">
        40.00
    </xu:update>
</xu:modifications>
```

A future Xindice development will likely include support for the W3C's XQuery language and XML Schema. It is likely to support full-text and schema-aware indexing.

2.3 XQuery

XQuery [46] is a query language that is designed to help retrieve data items from XML formatted documents. It is designed to query collections of XML data, not just XML files but anything that can appear as XML, including relational databases.

2.3.1 Structure of an XQuery Module

An XQuery module has three parts:

- Namespace and Schema Declarations (optional)
- Function Definitions (optional)
- Query Expressions

The first two are known as the prolog, the third part contains query expressions. Query expressions are the key to XQuery.

The principal forms of XQuery expressions are path expressions, element constructors, FLWOR expressions, list expressions, conditional expressions, quantified expressions, datatype expressions, and function calls.

Expressions are evaluated relative to a context namespaces, variables, functions, date and time, context item (current node or atomic value), context position (in the sequence being processed), and context size (of the sequence being processed).

Next we briefly introduce these expressions.

FLWOR expressions FLWOR expressions are the building blocks of XQuery. The name comes from the For, Let, Where, Order by, and Return keywords that make up the expression.

- The `for` clause provides a mechanism for iteration,
- The `let` clause allows variable assignments,
- The `for` and `let` clauses specify a sequence of tuples (a tuple is an ordered set of values).
- These tuples can then be filtered with a `where` clause and ordered using an `order by` clause.

- The `return` clause at the end of an expression indicates what should be returned. The `return` clause is evaluated once for every tuple surviving the `where` clause and ordered according to the `order by` clause. The return value of the FLWOR expression is the ordered sequence of content generated by the `return` clause as it evaluates each tuple.

Here is an example illustrating the use of a FLWOR expression:

```
for $i in document("data/items.xml")//item
  let $avg_price :=
     avg(document("data/items.xml")//item/price)
  where $i/price < $avg_price
  return    <LowPricedItems>
        $i/itemno, $i/description, $i/price
        </LowPricedItems>
```

Path expressions Path expressions are XPath 2.0 [45] expressions. They use a path notation to select nodes of interest from an XML document.

For example, the following path expression identifies all `item` elements that have a parent `items` element:

```
namespace ixq="http://www.ipedo.com/XQueryExample"
document("data/items.xml")/ixq:items/item
```

The following path expression finds all the `items` with the `itemType` "Purchasing item":

```
namespace ixq="http://www.ipedo.com/XQueryExample"
document("data/items.xml")/ixq:items/item[ItemType = "Purchasing item"]
```

Element constructors Element constructors are used to create new elements as part of the query output.

```
<newElement>Hello World</newElement>
```

The element constructor shown above constructs an element named `newElement`, that contains one child text node with the value "Hello World".

The following element constructor creates an element named "MyListOfExpensiveItems", containing as its child elements all item elements from the document items.xml that are priced over $1000.00.

```
namespace ixq="http://www.ipedo.com/XQueryExample"
<MyListOfExpensiveItems>
document("data/items.xml")/ixq:items/item[price > 1000]
</MyListOfExpensiveItems>
```

Conditional Expressions A conditional expression takes the form

```
if <condition_expr>
then <expr1>
else <expr2>
```

Quantified Expressions A quantified expression uses either an existential quantifier (some) or a universal quantifier (every). Every quantified expression evaluates to a Boolean value. The following examples demonstrate both kinds of quantifiers.

```
# returns purchase orders that only contain
# items cheaper than $300
namespace ns = "http://www.ipedo.com/XQueryExample"
for $i in document("data/PO.xml")/ns:polist/po
where every $p in
   $i/lineitems/lineitem/item/price
   satisfies ($p < 300)
return $i
```

The for clause returns all the purchase order elements in the document po.xml. The where clause uses a quantified expression to select only those purchase order elements where the price of every line item is less than $300.

```
#returns all purchase orders which contain at
#least one item priced more than $200
namespace ns = "http://www.ipedo.com/XQueryExample"
for $i in document("data/PO.xml")/ns:polist/po
where some $p in $i/lineitems/lineitem/item/price satisfies ($p > 200)
return $i
```

Like the previous query, the for clause returns all the purchase order elements in document po.xml. This time the where clause uses a quantified expression to select all the purchase orders that have at least one line item priced above $200.00.

2.3.2 XQuery Data Model

XQuery is defined in terms of a data model based on heterogeneous sequences of nodes and atomic values. The data model describes all the inputs to and the outputs from an XQuery processor. It also describes intermediate values. Value categories are:

- Sequence
  - Items
    - Nodes
      - Document
      - Element
      - Attribute
      - Text
      - Comment
      - Processing Instruction
      - Namespace
    - Atomic Values

An instance of the data model may contain one or more XML documents or fragments of documents. A query provides a mapping from one instance of the data model to another instance of the data model.

3 Querying in the Semantic Web

The Semantic Web [7] is an extension of the current web in which data is given an abstract representation, enabling it to be shared and processed by machines as well as by people. It is based on standards and technologies such as RDF Core Model [33], the RDF Schema [9] language and the Web Ontology Language [32]. These technologies all build on the foundation of URIs, XML, and XML namespaces. A major goal of the Semantic Web is to provide service discovery. Service descriptions is presented in formal languages or ontologies whose meaning is well defined and unambiguous. Given a set of service descriptions and a job specification written using a suitable collection of ontologies, a software agent takes care of selecting the appropriate service for the job and facilitates the interaction between the client and the service.

Next we present a short introduction to RDF and OWL together with query languages and tools that are designed to query information represented in these technologies.

3.1 Querying in RDF

The Resource Description Framework (RDF) [33] is a language for representing information about resources in the Web and expressing this information so it can be exchanged between applications without loss of meaning. In this section we present some query languages for RDF. First we give an overview of RDF.

3.1.1 The basic model

RDF is based on the idea of identifying things using Uniform Resource Identifiers (URIs), and describing resources in terms of simple properties and property values. This means that statements about resources are represented as a graph of nodes and arcs. For example, the statement "http://www.example.org/index.html has a creator whose value is John Smith" is represented by an RDF statement (triple) having:

- a subject http://www.example.org/index.html
- a predicate http://purl.org/dc/elements/1.1/creator
- an object http://www.example.org/staffid/85740

This statement can be represented by the graph shown in Figure 1.

RDF/XML [39] defines the XML syntax for such RDF graphs. For instance, the graph in Figure 1 can be represented in RDF/XML as:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:exterms="http://www.example.org/terms/">
  <rdf:Description rdf:about="http://www.example.org/index.html">
      <dc:creator rdf:resource="http://www.example.org/staffid/85740"/>
  </rdf:Description>
</rdf:RDF>
```

Statements describing application-specific resources and specific properties in describing those resources may need to define vocabularies (terms) that are not part of RDF. They can be described as an RDF vocabulary using RDF Schema.

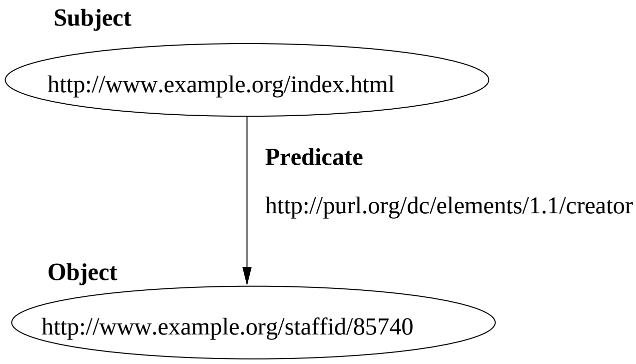

Figure 1: Basic RDF Model

3.1.2 RDF Query Languages

Although there is no one standard RDF query language, there are many RDF query languages [38] and there are many similarities among them. Many of these languages have different syntaxes but do basically the same thing, i.e., describe an RDF graph with parts missing, assign those parts variable names, and get a series of bindings to those variables. We present here some of such query languages. A comprehensive survey of RDF query languages is given in [38]. We first present the minimal requirements an RDF query must have.

Some of the requirements [37] for an RDF query language are:

- RDF graph pattern matching must be restricted to graph patterns, i.e., a query must satisfy one or more RDF triples
- Variable binding results for query must be zero or more. Each set of bindings is one way that the query can be satisfied by the queried graph.
- Extensible value testing must be possible, e.g., through function calls, namespace.
- Subgraph results must be possible to be returned as a result of the original queried graph.
- Local queries must be provided for accessing local RDF data, i.e., from the same machine or same system process.
- Optional match must be possible in the sense that if some specified part of the query fails to match the query does not fail.
- Limited datatype support must be included for a subset of XSD datatypes and operations on those datatypes.

Next, we briefly discuss some of RDF query languages.

rdfDB Query Language rdfDB [34] is a database intended to be a simple, scalable, open-source for RDF metadata. It uses a high-level, simple graph matching, triple-based SQL-like query language. This query language differs slightly in syntax from SquishQL [42]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. There is one query command which has the syntax

```
select [variable1, variable2, ... ... ] from {database} where
[constraint1, constraint2, ...] </>
```

which returns a set of variable bindings for `[variable1, variable2 ... ]` such that the triples in `[database]` satisfy `[constraint1, constraint2, ... ]` under those variable substitutions.

The query

```
select ?x from people where (worksFor ?x RISC) (name ?x ?y) </>
```

returns, from the `people` database, all persons who `worksFor` RISC.

Algae [2] is a query language that provides query and assertion access to RDF graphs. It is designed to query a graph, insert data into a graph, or write rules to automatically insert data when a query is matched. One of its main uses is with the Annotea [3] which is a protocol that enhances collaboration via shared metadata based Web annotations (such as comments, notes, explanations, or other types of external remarks that can be attached to any Web document), bookmarks, and their combinations. The following Algae query shows how to select nodes from and RDF Graph.

```
ask (?x <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
           <http://www.w3.org/2000/10/annotation-ns#Annotation>)
```

The query binds the variable `?x` to each node with `type Annotation`. For example, `?x` will bind to `Annot1` and `Annot8` in the sample data

```
<r:RDF xmlns:r="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
       xmlns:a="http://www.w3.org/2000/10/annotation-ns#">
  <r:Description r:about="http://example.com/things#Annot1">
    <r:type
     r:resource="http://www.w3.org/2000/10/annotation-ns#Annotation"/>
  </r:Description>
  <a:Annotation r:about="http://example.com/things#Annot8" />
<r:RDF>
```

SquishQL [42] is an SQL-like query language for RDF. It supports a query model based on a graph pattern formed from variables for nodes, arcs and literals. Additionally it uses filter functions in the form of Boolean expressions over the variables to restrict their values. Here is an Squish query to perform text matching:

```
SELECT ?title, ?description
FROM http://test1/test, http://test2/test
WHERE
(dc::creator ?doc ?sname)
(dc::title ?doc ?title)
(dc::description ?doc ?description)
AND ?sname ~ brickley
USING dc FOR http://purl.org/dc/1.1/
```

RDQL [40] is a language that evolved from several query languages specially from SquishQL. Its purpose is to extract information from an RDF graph by treating RDF as data and providing query with triple patterns (each triple pattern is made of named variables and RDF values) and a set of constraints on the values of the variables. The triple pattern

```
SELECT ?x
WHERE (?x,  <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>,
                                  <http://example.com/someType>)
```

matches all statements in the graph that have predicate [`http://www.w3.org/1999/02/22-rdf-syntax-ns#type`] and object [`http://example.com/someType`]. The variable `?x` will be bound to the label of the subject resource.

RDQL is used as the query language for RDF in Jena [23] models. In this case, it provides a data-oriented (there is no inference being done. Jena model gives the impression that required triples exist by creating them on-demand) query model so that there is a more declarative approach to complement the fine-grained, procedural Jena API. RDQL only takes the description of what the application wants, in the form of a query, and returns that information, in the form of a set of bindings.

RDFQL [36] is an SQL-like language that is used by RDF Gateway [35] applications for querying RDF data models. "Agents" that submit queries to RDF Gateway service use RDFQL to describe the information they need as a `query condition` and it is processed by by the service to find documents on the Internet containing the exact information described by it. RDFQL uses SQL commands such as `INSERT`, `DELETE` and `SELECT` to perform query operations on RDF triples, and data definition commands such as `CREATE TABLE` or `CREATE VIEW`. It uses the `USING` clause, instead of the `FROM` clause, to specify the data sources to use. It also supports complex query conditions, defined in the `WHERE` clause, by using most of the SQL functions and operators. The `ORDER BY` clause provides ways to order the results.

3.2 Querying in OWL

The Web Ontology Language (OWL) [32] builds on top of RDF and RDF Schema with greater machine interpretability of data. It is designed to be used by applications that need not only present but also process the content of information. It provides additional vocabularies along with their formal semantics. OWL has three sublanguages:

- OWL Lite – uses only some of the OWL language features. It is used in describing a classification hierarchy and simple constraints.
- OWL DL – provides maximum expressiveness while conclusions are guaranteed to be computable in finite time. It includes all OWL language constructs, but they can be used only under certain restrictions. OWL DL is so named due to its correspondence with Description Logics, a field of research that has studied the logics that form the formal foundation of OWL.
- OWL Full – provides maximum expressiveness and the syntactic freedom of RDF with no computational guarantees.

3.2.1 OWL Query Languages

Many query languages and tools presented in section 3.1.2 can be used as a basis to query documents presented in OWL. For instance, 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).

OWL Query Language (OWL-QL) [18] is intended to be a candidate standard language and protocol for query-answering dialogues among Semantic Web computational agents (answering agents (servers) derive answers to question posed by querying agents (clients)) using knowledge represented in OWL. It specifies the semantic relationships among a query, a query answer, and the knowledge base. The knowledge base may comprise multiple knowledge bases on the Semantic Web that are not necessarily specified by the client.

A simple OWL-QL query-answering dialogue is initiated by a client sending a query to an OWL-QL server. The `query pattern` specifies a collection of OWL statements with some `URIrefs` as variables. For example, the query pattern shown below can be used to ask "Who owns a red car?".

```
Query: (``Who owns a red car?'')
 Query Pattern: {(owns ?p ?c) (type ?c Car) (has-color ?c Red)}
 Must-Bind Variables List: (?p)
 May-Bind Variables List: ()
 Don't-Bind Variables List: ()
 Answer Pattern: {(owns ?p  a red car )}
 Answer KB Pattern: ...

Answer: ( Joe owns a red car? )
 Answer Pattern Instance: {(owns Joe  a red car )}
 Query: ...
 Server: ...
```

A variable may be a `must-bind`, a `may-bind`, or a `don't-bind` variable. The answer binds literals to variables in the query pattern and is entailed by the answer knowledge base. An answer is required to provide bindings at least for the `must-bind` variables. OWL-QL allows assumptions in a query by means of "if-then". It allows a query to have a `query premise` which is an OWL knowledge base reference. The query

```
Query: ``If C1 is a Seafood Course and W1 is a drink of C1, then what
color is W1?''
Premise: {(type C1 Seafood-Course) (has-drink W1 C1)}
Query Pattern: {(has-color W1 ?x)}
Must-Bind Variables List: (?x)
...
```

is an example of a query that has a premise.

4 Querying in MKM-NET

The Mathematical Knowledge Management Network (MKM-NET) [28] aims at finding ways to manage mathematical knowledge. Among its objectives is mathematical knowledg representation and searchability.

4.1 Searching Directions in MKM-NET

The MKM-NET document, Mathematical Knowledge Management and Searchability [12], describes the state of the art for searching mathematical content. It states the following approaches for searching mathematical content ranging from basic textual methods to semantic-driven techniques.

- Fulltext Search – This method has its limitation, namely, when parts of the content consist of formulas. To overcome this, search interfaces has to be augmented with tools, such as computer algebra and Latex shells, to enable the user to enter a formula description taking into account different formula encodings.

- Semi-textual search – If the structure of documents is more regular than that of the everyday mathematical notation, i.e., semi-natural, then using fulltext search tools such as `grep` can be very effective. For example, a retrieval system has been integrated with `grep` as part of the Mizar project [27]. Mizar Mathematical Library (MML) is a comprehensive library of formal mathematical knowledge that is expressed by means of semi-natural language. It has a set of primitive operations returning, for instance, for each article the set of concepts introduced in it, or for each mathematical item the set of all other items which are referenced in it, or which use it). Operations over lists, such as union, merge, filter and so on allow the construction of compound queries. The main filtering operations is still textual, directly calling `grep`.

- Semantic Search for Terms – Mathematical text often has an agreed upon standard semantics that can be exploited to improve search. Open Mathematical Documents (OMDoc) [31], which is an extension of OpenMath supports searching by taking advantage of standard semantics. Its basic idea is to specify the way in which mathematical text is augmented by a description of its meaning. MBase [24] can be seen as a set of collections of OMDocs and it supports the semantic retrieval of contents. ActiveMath [1] system uses a technique to link each occurrence of a symbol in a document to its definition. This technique allows the user to make a search template with relevant semantic information. When search is restricted to selected content objects only, these objects may be presented in a document as links. These links form a further search for specific content. Adding semantic annotation makes it possible to retrieve template instances. For example a search for "$X \otimes Y \otimes X$", where X and Y are variables (wildcards) in the template, will retrieve $A \otimes B \otimes A$ as well as $A \otimes (B \otimes C) \otimes A$ and $B \otimes A \otimes B$ but not $A \otimes B \otimes C$.

- Fast Filtering of the Search Space – Apply a unification process externally on a set of candidate matches and rapidly filter the resulting data order to produce a small set of candidates. To perform this, HELM [20] proposed an approach based on four distinct phases:
  - Data-mining: automatically computing a small set of metadata from each theorem or definition in the library to provide an estimate of its actual content.
  - Pattern compilation: when a semantic query is submitted, it is compiled into a low-level query.
  - Filtering: the resulting low-level query is executed over the database of metadata, obtaining a set of candidates.
  - Matching: the actual semantic query is iterated over all candidates, obtaining a precise result set.

- Semantic Search for Knowledge – Many natural language sentences in mathematical documents have a precise meaning. For example the sentence "if A and B are groups, then $A \otimes B \otimes A$ is a group too" can be formalized as $group(A) \wedge group(B) \Rightarrow group(A \otimes B \otimes A)$. Thus searching for the template $X \Rightarrow group(Y \otimes Z \otimes U)$ would yield the sample sentence in the database that A and B being groups is a sufficient condition for $A \otimes B \otimes A$ being a group.

- Key Phrase Search – To evaluate the relevance of the occurrences of some mathematical content returned by a search using automated indexing techniques.

- Exploiting Semantic Relation – To deal with content that is related to other content. For example, Theorems have proofs which make use of other theorems; exercises require understanding of previous theorems etc. This gives the possibility of composing meaningful documents for specific purposes. What makes this easy is the fact that semantic relations between pieces of content are binary.

- Combination – Combining semantic relations with key phrase search and types of content by means of an automated inference engine may help solving the problem of ensuring the relevance of the mathematical objects retrieved.

In the following, we describe querying facilities used in some MKM-NET projects.

4.2 Searching in MBase

Mathematical Knowledge Base (MBase) [24] is a specific database (a set of collections of OMDocs) supporting the semantic retrieval of terms. Currently MBase has the following facilities (implemented as Web interface) for retrieving content from the database:

- QuickSearch finds a text fragment in any mathematical element. For example, a search for the text fragment "bool" would return as a result collections (e.g., logics, OMCDs), theories, and items where the text fragment occurs.

- Pattern Search is used for structural queries on the set of stored terms and formulas. for example, searching for the pattern "eq(a(F X Y) a(F Y X))" (which represents the commutativity f(x,y) = f(y,x)) would return as a result all found matching terms.

- Specific Elements Search is used for specific math elements such as Theory, Symbol, Definition, Axiom, and Theorem.

Although it provides these search and query facilities, MBase still does not have a specialized query language yet. The future query requirements MBase is set to support are:

- General SQL-like queries based on the structure of the OMDoc input.

- Queries which match formulae under a given equality theory. An example is:

  ```
  If a + (b + c) is in MBase, then also (b + c) + a and (a + b) + c
  should be found.
  ```

- Queries with meta-variables. An examples is:

```
Give me at least one lemma of the form
        Forall x.G(x) <= f(x) <= H(x)

 where G and H should be some Meta-Variables which match some term
 in x. f(x) is a given term, e.g. sin(x).
```

4.3 Information Retrieval in Mizar

The main activity of the Mizar project [27] is the development of a database for mathematics called the Mizar Mathematical Library (MML) [4]. MML consists of Mizar Articles. Mizar also has developed a proof-checking system called the Mizar System which is the implementation of the Mizar Language. Information Retrieval in MML amounts to searching, browsing, and presentation of content. Mainly it provides:

- HTML based browsing.
- Text based searching which uses tools from the `grep` family and includes searching for theorems and searching for definitions.
- Semantic based retrieval which makes use of operations on `resources`. These are pieces of information stored in internal MML files such as `constructors`, `patterns`, `statements`, `theorems`, etc. The following query returns the list of resources where both the listed constructors occur ("& occur") and then filters all theorems from the list ("— th").

  ```
  {XBOOLE_0:func 1, XBOOLE_0:func 3} & occur | th
  ```

4.4 Searching and Retrieving in HELM

The Hypertextual Electronic Library of Mathematics (HELM) project [20] uses XML for the creation and maintenance of a virtual, distributed, hypertextual library of formal mathematical knowledge. The main emphasis of HELM is on rendering, browsing, management, searching and retrieving issues. It is more oriented towards proof assistant applications than computer algebra systems. It uses Coq proof assistant [11] and MathML [26] as essential components. HELM distinguishes between two kinds of basic queries:

- Not specific to mathematical domain and uses information such as names, keywords or authors. An example query is "Search and retrieve all the theories in which a given knowledge item is used/referenced".
- Specific to mathematical domain (such as proof-searching). It allows to search for objects including terms satisfying some given constraints expressed as match patterns on types. Examples:
  - Search and retrieve all the theories regarding Linear Algebra.
  - Search and retrieve all the theorems that are applicable to a given set of hypotheses.
  - Search and retrieve all the theorems whose conclusion matches a given type pattern.
  - Search and retrieve all the proofs of a given statement.

The first kind is implemented in MathQL-1 (see below). The second kind of search is a work under progress (MathQL-1, MathQL-2, and MathQL-3). It is seen as expensive and complex. HELM uses metadata to help this kind of search by associating to every theory, collection of objects or a single object an XML file (a metadata model) with the relative meta-information. Metadata model can be expressed in RDF. RDF metadata can be queried using two approaches:

- Inserting RDF metadata in a relational or XML database and use SQL for querying.
- Treating RDF metadata as a knowledge base an apply knowledge representation and reasoning techniques.

HELM has an MathQL proposal which aims at the development of a set of query languages:

- MathQL-1 is focused on querying an arbitrary RDF database. The peculiar aspects of this language concern the query results.
- MathQL-2 will include content-based pattern-matching.
- MathQL-3 will include other forms of formal matching involving for instance isomorphism, unification and definitions expansion.

MathQL-1 has facilities for hierarchical constraints based on RDF Schema and for traversing the compound values of properties. It particularly concerns the query results. Query results are treated as attributed values that can exist as data stored in the set of the subject strings, in attributes in each group, in different groups and in the set of strings stored in attribute values. The language has a defined semantics, both for queries and for query results. The Query engine is written in Caml (a strongly-typed functional language) and consists of the following components:

- The basic Caml package for MathQL-1 which provides a representation of queries and query results.
- The interpreter (mathql_interpreter) which provides the proper search engine. It executes a query given in Caml representation and gives back a Caml representation of the query result.
- The query generator (mathql_generator) is the interface between the interpreter and the HELM proof assistant.

4.5 Querying in MONET

MONET [29] defines a set of ontologies to model service descriptions (which are basically ontological conversions of MSDL [13, 30] descriptions) as well as queries on those descriptions. These ontologies are, written in OWL [32], used by a component within the MONET architecture called "Instance Store" [21] which, by using a "Description Logic" reasoner called RACER [19], matches queries to appropriate services. Next we give an overview of these ontologies and then describe how they are used to transform MSDL description into owl and how these transformed owl description are being matched against.

MONET uses two classes of ontologies: those describe models internal to MONET (e.g., problem and software) and those describe models external to MONET (e.g., OpenMath and GAMS). Individual ontologies of both classes are imported into one MONET ontology.

4.5.1 MONET Ontologies

Monet has the following ontologies:

- GAMS ontology is a simple class hierarchy; each GAMS class corresponds to an OWL Class, and specialization of a problem class is represented by a sub-class relationship.
- Symbolic ontology is an extension of the GAMS ontology to add symbolic computation category.
- OpenMath ontology has one root class called `OpenMathSymbol`. It has subclasses corresponding to symbols defined in a particular content dictionary. The symbols themselves are defined as further subclasses.
- Hardware ontology has two classes; `Computer` and `Manufacturer`. For instance a `SharedMemory` machine is manufactured by `Sun`.
- Software ontology describes pieces of software. It is intended to be used in the implementation part of the service description.
- Problems ontology represents individual problems as a classes which can have properties indicating bibliography entries and generalizations or specializations of them.
- Algorithms ontology represents elements of the algorithm library in the MONET architecture. It has two subclasses; Algorithm and Complexity.
- Directives ontology is a collection of classes which identify the task that is performed by the service.
- Theory ontology represents available formalized theories in digital libraries of mathematics.
- Bibliography ontology represents entries in bibliographic indexes such as Zentralblatt MATH [48] to be associated with algorithms.
- Encodings ontology represents the formats used for encoding mathematical objects in the MONET framework.
- MONET ontology imports all the ontologies described above and is used to represent complete service descriptions and queries.

When a service is submitted to the MONET broker, its description is presented in MSDL. This description is transformed into OWL Abstract Syntax [6] by means of an XSLT stylesheet written for this purpose. Service matching is then performed by submitting a query to the Instance store in the form of OWL description. Instance Store answers the queries by using a combination of Description Logic (DL) reasoning and database queries.

The following query asks for all individuals whose GAMS classification is `GamsG`. The Instance Store answers the query by returning `nagopt` and `nagopt-variation`.

```
restriction(
     <http://monet.nag.co.uk/owl#service_classification>
     someValuesFrom(
        restriction(
                 <http://monet.nag.co.uk/owl#gams_class>
```

```
                  someValuesFrom(<http://gams.nist.gov#GamsG>)
) ) )
```

The following query asks for all individuals with `algorithm` having a `Zentral Blatt` bibliographic reference `0277.65028`. The answer is `nagroot`.

```
restriction(
 <http://monet.nag.co.uk/owl#service_implementation>
  someValuesFrom(
   restriction(
    <http://monet.nag.co.uk/owl#service_algorithm>
    someValuesFrom(
      restriction(
       <http://monet.nag.co.uk/algorithm#bibliographic_reference>
       someValuesFrom(
        intersectionOf(
         <http://monet.nag.co.uk/bibliography#ZentralBlatt_MATH>
         restriction(
          <http://monet.nag.co.uk/bibliography#bibref>
          value("Zbl_0277.65028"^^<http://www.w3.org/2001/XMLSchema#string>)
) ) ) ) ) ) ) )
```

5 Querying in MathBroker

The way MSDL descriptions in MathBroker are published influences the way they can be queried. Currently MSDL descriptions are represented in the MathBroker registry as repository items in the following way: we extended the ebXML registry information model such that each MSDL information entity is an extension of the `ExtrinsicObject` entity of ebXML registry. In this case an MSDL entity is presented in the registry by a minimal metadata (such as name, uuid, description, classification, and association). The rest of the details are stored in the respective repository item of the entity. This repository item is stored in the registry file system (called repository). Querying in this case can be done at two levels:

- Registry-level querying– which is supported by the registry's underlying query capabilities (see Section 2.1) such as querying by name, by id, and by classification depending on the metadata submitted to the registry. This kind of querying has been implemented in the MathBroker registry [5].

- MSDL-level querying– this will involve the repository item stored in the registry. The repository is in MSDL syntax which is basically a schema-based XML document. Querying here would involve query languages devised for XML-based documents, Semantic Web, and MKM. Our goal is to develop a query language that facilitates this type of querying.

6 Conclusion

In the previous sections, we have briefly presented a set of query languages and tools for querying specific information representation formalisms, namely XML, Semantic Web, and Mathematical Knowledge Management. We presented them according to the level of expressiveness from simple query languages for basic XML node structures to those used to query resource description documents in RDF and ontological structures in OWL and then more specialized languages that are used to query mathematical documents. Query languages for XML-like documents are matured enough to support a complete set of modeling constructs offered by such documents. Query languages for the Semantic Web do not yet support the complete set of modeling constructs offered by Semantic Web standards such as RDF, RDF Schema, and OWL. Query languages for MKM follow one of the mentioned query languages because the mathematical documents they support belong to one of the above standards or at least resembles it in its representation. These languages are still far from being able to support the highly structured nature of mathematical documents.

References

[1] The ActiveMath Learning Environment, September 2004. http://www.activemath.org/.

[2] Algae RDF Query Language. W3C, June 2004. http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/.

[3] Annotea Project. W3C, January 2004. http://www.w3.org/2001/Annotea/.

[4] Grzegorz Bancerek and P. Rudnicki. Information Retrieval in MML. Proceedings of the Second International conference on Mathematical Knowledge Management, Bertinoro, Italy, February 2003. LNCS 2594, pp119-131.

[5] Rebhi Baraka, Olga Caprotti, and Wolfgang Schreiner. A Registry Service as a Foundation for Brokering Mathematical Services. Technical report, RISC-Linz, Austria, February 2004. Available from ftp://ftp.risc.uni-linz.ac.at/pub/techreports/2004/04-13.ps.gz.

[6] Sean Bechhofer, Peter F. Patel-Schneider, and Daniele Turi. OWL Web Ontology Language Concrete Abstract Syntax. Technical report, The University of Manchester, UK, December 2003. Available from http://owl.man.ac.uk/2003/concrete/latest/.

[7] Tim Berners-Lee, James Hendler, and Lassila Ora. Semantic Web. Scientific American, May 2001. Available from http://www.sciam.com/2001/0501berners-lee.html.

[8] Tim Bray et al. Extensible Markup Language (XML) 1.0 (Third Edition). W3C Recommendation, February 2004. available from http://www.w3.org/TR/2004/REC-xml-20040204/.

[9] Dan Brichley and R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation, February 2004. Available from http://www.w3.org/TR/rdf-schema/.

[10] Olga Caprotti and Wolfgang Schreiner. Towards a Mathematical Service Description Language. In *International Congress of Mathematical Software ICMS 2002*, Bejing, China, August 17–19, 2002. World Scientific Publishing, Singapore.

[11] The Coq Proof Assistant, September 2004. http://coq.inria.fr/.

[12] I. Dahn and A. Asperti. Mathematical Knowledge Management and Searchability. Deliverable D5.4, 2001. Available from http://monet.nag.co.uk/mkm/MKMNetTN-D5-4.pdf.

[13] Mike Dewar, David Carlisle, and Olga Caprotti. Description Schemes for Mathematical Web Services. In *EuroWeb 2002: The Web and the Grid: From e-Science to e-Business*, Oxford, UK, December 2002. Britisch Computer Society Electronic Workshops in Computing.

[14] Document Object Model (DOM). W3C, September 2004. http://www.w3.org/DOM/.

[15] ebXML Registry Information Model Schema, April 2002. http://www.oasis-open.org/committees/regrep/documents/2.1/schema/rim.xsd.

[16] ebXML Registry Information Model v2.0. OASIS, December 2001. http://www.oasis-open.org/committees/regrep/documents/2.0/specs/ebRIM.pdf.

[17] ebXML Registry Services Specification v2.0. OASIS, April 2002. http://www.oasis-open.org/committees/regrep/documents/2.0/specs/ebRS.pdf.

[18] Richard Fikes, Patrick Hayes, and Ian Horrocks. OWL–QL – A Language for Deductive Query Answering on the Semantic Web. Knowledge Systems Laboratory, 2003. Available from http://ksl.stanford.edu/KSL_Abstracts/KSL-03-14.html.

[19] Volker Haarslev and Ralf Moller. Description of the RACER system and its applications. In *Automated reasoning: First International Joint Conference, IJCAR 2001*, Siena, Itally, June 18–23, 2001. volume 2083 of Lecture Notes in Artificial Intelegence, New York, NY, USA, 2001. Springer Verlag Inc.

[20] Helm: Hypertextual Electronic Library of Mathematics, September 2004. http://helm.cs.unibo.it/.

[21] Instance Store - database support for reasoning over individuals. The University of Manchester, 2002. http://instancestore.man.ac.uk/instancestore.pdf.

[22] Database Language SQL Part 4: Persistent Stored Modules (SQL/PSM). ISO, December 1996.

[23] Jena - A Semantic Web Framework for Java. Sourceforge, September 2004. http://jena.sourceforge.net/.

[24] M. Kohlhase and H. Franke. MBase: Representing Knowledge and Content for the Integration of Mathematical Software Systems. Journal of Symbolic Computation, 32(4), pp365-402, 2001.

[25] MathBroker — A Framework for Brokering Distributed Mathematical Services. Reseach Institute for Symbolic Computation (RISC), April 2004. http://www.risc.uni-linz.ac.at/projects/basic/mathbroker.

[26] Mathematical Markup Language (MathML) Version 2.0. W3C Recomendation, February 2001. http://www.w3.org/TR/MathML2.

[27] Mizar Project, September 2004. http://mizar.org/project/.

[28] Mathematical Knowledge Management Network, November 2003. http://monet.nag.co.uk/mkm/index.html.

[29] MONET — Mathematics on the Web. The MONET Consortium, April 2004. http://monet.nag.co.uk.

[30] Mathematical Services Description Language (MSDL). Reseach Institute for Symbolic Computation (RISC), April 2004. http://poseidon.risc.uni-linz.ac.at:8080/mathbroker/results/xsd.html.

[31] OMDoc: A Standard for Open Mathematical Documents, September 2004. http://www.mathweb.org/omdoc/.

[32] OWL Web Ontology Language Reference. W3C, February 2004. available from http://www.w3.org/TR/2004/REC-owl-ref-20040210/.

[33] Resource Description Framework (RDF). W3C, September 2004. http://www.w3.org/RDF/.

[34] Annotea Project. W3C, January 2004. http://guha.com/rdfdb/.

[35] RDF Gateway a Platform for Semantic Web. Intellidimension, September 2004. http://www.intellidimension.com/.

[36] RDFQL. Intellidimension, September 2004. http://www.intellidimension.com/.

[37] RDF Data Access Use Cases and Requirements. W3C, August 2004. http://www.w3.org/TR/2004/WD-rdf-dawg-uc-20040802/.

[38] RDF Query Survey. W3C, April 2004. http://www.w3.org/2001/11/13-RDF-Query-Rules/.

[39] RDF/XML Syntax Specification. W3C, February 2004. http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/.

[40] RDQL - A Query Language for RDF. W3C, January 2004. http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/.

[41] Structured Query Language. FIPS, June 1993. http://www.itl.nist.gov/fipspubs/fip127-2.htm.

[42] RDF Squish Query Language. ILRT, February 2001. http://ilrt.org/discovery/2001/02/squish/.

[43] Apache Xalan. The Apache XML Project, September 2004. http://xml.apache.org/xalan-j/.

[44] Apache Xindice. The Apache XML Project, September 2004. http://forrestbot.cocoondev.org/sites/xml-xindice/index.html.

[45] XML Path Language (XPath) 2.0. W3C, July 2004. http://www.w3.org/TR/xpath20/.

[46] XQuery 1.0: An XML Query Language. W3C, July 2004. http://www.w3.org/TR/xquery/.

[47] XML Update Language (XUpdate). XML:DB, September 2004. http://xmldb-org.sourceforge.net/xupdate/.

[48] Zentralblatt MATH, September 2004. http://www.emis.de/ZMATH/.