Publishing and Discovering Mathematical Service Descriptions: A Web Registry Approach

Rebhi Baraka, Olga Caprotti, and Wolfgang Schreiner*

Research Institute for Symbolic Computation (RISC-Linz)
Johannes Kepler University, Linz, Austria
{rbaraka,ocaprott,schreine}@risc.uni-linz.ac.at

Abstract. We describe an approach for publishing and discovering mathematical service descriptions in a Web registry. The registry is based on and extends the ebXML registry to handle descriptions given in the Mathematical Service Description Language MSDL. This work is a first step towards our ultimate goal to produce a "semantic broker" where services register their problem solving capabilities and clients submit task descriptions; the broker then determines the suitable services and returns them to the client for invocation.

1 Introduction

The interest of the mathematical community to deploy its software over the Internet has paved the way for the emergence of mathematical Web services. WebMathematica [18] and MapleNET [11] are commercial approaches towards this goal; MathWeb [15] is a long-term academic activity. Open technologies such as OpenMath [4] and MathML [1] are employed to accomplish the exchange and presentation of mathematical objects in the Web.

A Web service is a problem solution that is available on the Web and can be accessed by a user or another service or program via standard Web protocols [19]. A mathematical Web service is a service that offers the solution to a mathematical problem (based on e.g. a computer algebra system or on an automated theorem prover). Web services need to be advertised by providers and discovered by clients; therefore they need to be described in a machine-understandable format. In the case of mathematical Web services, these descriptions must be based on formal mathematics.

There have been several approaches to achieving this goal, in particular MONET [16] and our own MathBroker project [12], which have been pursued simultaneously with mutual influence. MathBroker uses the concept of a Web registry for publishing and discovering descriptions of mathematical Web services. A (mathematical) registry provides a set of functionalities to facilitate the sharing and exchange of (mathematical) service descriptions. For this purpose, we developed the Mathematical Services Description Language (MSDL)

* This work was sponsored by the FWF Project P15183 "A Framework for Brokering Distributed Mathematical Services".

[3] and extended an existing registry implementation to handle the publication and discovery of objects that are presented in MSDL descriptions [6, 5].

The remainder of this paper is organized as follows: Section 2 introduces our model for describing mathematical services; this represents the basis for the design of MSDL and for an extension of the information model of a widely used registry framework. Section 3 describes this extension in greater detail. In Section 4, we present a sample client for using the extended framework for publishing MSDL descriptions and discovering such descriptions.

2 A Model for Mathematical Service Descriptions

Figure 1 illustrates our model for the description of mathematical Web services. It shows the kinds of entities that can be associated to a service and the relationships among them.

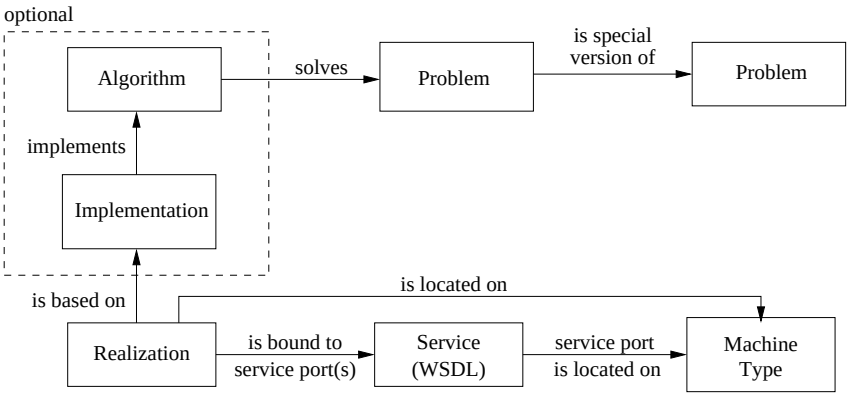

Fig. 1. Service Description Model

The model is implemented as a highly structured language called Mathematical Services Description Language (MSDL) [14]. MSDL was developed in the frame of MathBroker project [12] with influences from the MONET project [16].

The rationale behind the decomposition of descriptions into multiple interlinked entities is to avoid redundancy between specifications (by sharing description components) and to provide a quick shortcut for detecting the identity of specification entities by reference equality. MSDL thus provides for the development of a reusable library of descriptions.

Now we introduce the entities of the model and give sample descriptions in MSDL syntax which conforms to a grammar of an MSDL schema [3].

Problem: There exist various kinds of problems. For instance, a computing problem can be specified by input parameters, an input condition, output parameters, an output condition. It can be a special version of another

problem. A problem description for the indefinite integration of a function $f : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ may look like:

```
<monet:problem name="indefinite-integration">
    <monet:header></monet:header>
    <monet:body>
      <monet:input name="f">
        <monet:signature>
          <om:OMOBJ>
            <om:OMA>
              <om:OMS cd="sts" name="mapsto"></om:OMS>
              <om:OMS cd="setname1" name="R"></om:OMS>
              <om:OMS cd="setname1" name="R"></om:OMS>
              <om:OMS cd="setname1" name="R"></om:OMS>
            </om:OMA>
          </om:OMOBJ>
        </monet:signature>
      </monet:input>
      <monet:output name="i">
        <monet:signature>
          <om:OMOBJ>
            <om:OMA>
              <om:OMS cd="sts" name="mapsto"></om:OMS>
              <om:OMS cd="setname1" name="R"></om:OMS>
              <om:OMS cd="setname1" name="R"></om:OMS>
              <om:OMS cd="setname1" name="R"></om:OMS>
            </om:OMA>
          </om:OMOBJ>
        </monet:signature>
      </monet:output>
      <monet:post-condition>
        <om:OMOBJ>
          <om:OMA>
            <om:OMS cd="relation1" name="eq"></om:OMS>
            <om:OMV name="i"></om:OMV>
            <om:OMA>
              <om:OMS cd="calculus1" name="indefint"></om:OMS>
              <om:OMV name="f"></om:OMV>
            </om:OMA>
          </om:OMA>
        </om:OMOBJ>
      </monet:post-condition>
    </monet:body>
  </monet:problem>
```

Algorithm: an algorithm is described by (a link to the description of) the problem it solves, as well as by time and memory complexity, termination conditions, and bibliographical information.

```
<monet:algorithm name="RischAlg">
  <monet:documentation>
    The Risch algorithm for indefinite integration.
  </monet:documentation>
  <monet:problem href="http://risc.uni-linz.ac.at/
     mathbroker/RischIndefIntegration/indefinite-integration">
  </monet:problem>
  <monet:bibliography href="http://www.emis.de/cgi-bin/zmen/
     ZMATH/en/quick.html?type=xml&an=0184.06702">
    <!-- more dublin core -->
    <monet:documentation> Dublin Core
      Data</monet:documentation>
    <dc:creator>Risch,R.H.</dc:creator>
    <dc:title>The Problem of Integration in
       Finite Terms</dc:title>
    <dc:source>Trans. A.M.S. 139 pp.167 - 189</dc:source>
    <dc:publisher>AMS</dc:publisher>
    <dc:date>1969</dc:date>
  </monet:bibliography>
</monet:algorithm>
```

Implementation: an implementation is described by the algorithm on which it is based (or optionally the problem it solves), the software on which it is based, time and memory efficiency w.r.t. some reference architecture.

```
<monet:implementation name="RImpl">
  <mathb:efficiency_factor wrt="S200Spec">
    <mathb:speed>1.1</mathb:speed>
    <mathb:throughput>0.7</mathb:throughput>
  </mathb:efficiency_factor>
  <monet:software href="http://www.wolfram.com">
  </monet:software>
  <monet:software href="http://riaca.win.tue.nl/software/
    ROML"></monet:software>
  <monet:hardware href="http://risc.uni-linz.ac.at/mathbroker/
    RischIndefIntegration/perseus.risc.uni-linz.ac.at">
  </monet:hardware>
  <monet:algorithm href="http://risc.uni-linz.ac.at/
    mathbroker/RischIndefIntegration/RischAlg">
  </monet:algorithm>
</monet:implementation>
```

Realization: a realization of a service is described by the underlying software implementation (or optionally the algorithm or problem), by the type of

machine on which it is running and by a WSDL description of the service interface.

```
<monet:service name="RRISC">
  <monet:documentation>
    This is an implementation of the Risch algorithm.
  </monet:documentation>
  <monet:classification>
    <monet:problem href="http://risc.uni-linz.ac.at/
      mathbroker/RischIndefIntegration/
      indefinite-integration">
    </monet:problem>
  </monet:classification>
  <monet:implementation href="http://risc.uni-linz.ac.at/
    mathbroker/RischIndefIntegration/RImpl">
  </monet:implementation>
  <monet:service-interface-description
    href="http://perseus.risc.uni-linz.ac.at:8080/axis/
    services/SymbolicIntegration?wsdl">
  </monet:service-interface-description>
  <monet:service-binding>
    <monet:map action="exec"
      operation="symbint:Integrator:indefInt"
      problem-reference="indefinite-integration"></monet:map>
    <monet:message-construction io-ref="f"
      message-name="symbint:IndefIntRequest"
      message-part="in0">
    </monet:message-construction>
  </monet:service-binding>
  <monet:service-metadata></monet:service-metadata>
  <monet:broker-interface>
    <monet:service-URI></monet:service-URI>
  </monet:broker-interface>
</monet:service>
```

We skip the description of the machine and of the service interface.

For processing MSDL descriptions, a Java API was designed and implemented (see Section 3.6).

3 A Registry for the Service Description Model

In Web technology, a registry is a service for publishing and discovering information about Web services. A registry maintains Web service metadata as objects in a repository and provides access to them via a specific protocol. Currently there are two dominating registry standards:

- The Universal Description, Discovery, and Integration (UDDI) registry [17].
- The ebXML registry and repository standard [9].

We based our development on the ebXML registry reference implementation [8] because its information model [7] is much more generic and extensible than UDDI. Furthermore, this model closely follows Sun's Java API for XML registries (JAXR) [10] which provides generic access a variety of XML registries.

3.1 The ebXML Registry

The ebXML registry architecture consists of a service and a client. The registry service manages the objects associated with the registry and queries for them. A registry client is an application that accesses the registry. It utilizes the registry service to submit objects, to classify them, to associate them to each other, to browse them, and to query for them.

The registry information model [7] represents a blueprint for the registry. It provides the information on the classes of metadata that are stored in the registry as well as the relationships among metadata classes. It defines what types of objects are stored in the registry and how they are organized. The information model is extensible to new kinds of objects.

3.2 Extending the Registry to Mathematical Service Descriptions

We extended the ebXML information model to accommodate the entities of our mathematical service description model. Figure 2 shows the corresponding Java classes and their relationships to each other.

A generic "MathbrokerObject" class was introduced as an extension of the ebXML class "ExtrinsicObject". From this class, all other MathBroker classes are inherited such that ebXML treats the MathBroker objects as instances of ExtrinsicObject. We do not have a class "Service" as in our information model; its information (a WSDL description) is directly included in "Realization".

The rationale for using ExtrinsicObject as the basis of our classes is that it allows to hold metadata about which the registry has no prior knowledge. In particular, it may hold an XML document, e.g., the MSDL description of the entity. The MathBroker subclasses provide additional methods to extract and to modify this information.

3.3 Association of Entities

An essential characteristic of our service description model is the ability to express relationships among the various entities that comprise a service. The registry facilitates this feature by the concept of an "Association". A registry object may be associated with zero or more other registry objects. The service description model defines the following mathematical associations (see Figure 2):

IsSpecialVersionOf: Problem P "is special version of" Problem P'.

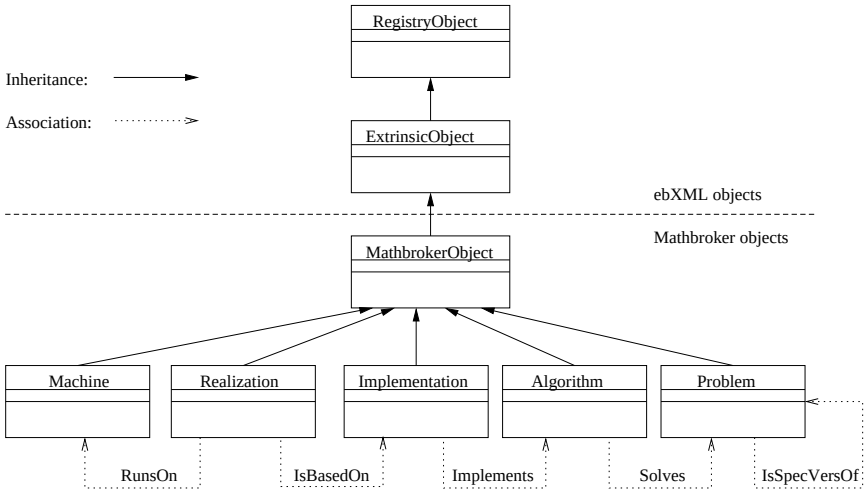

Fig. 2. Service Description Model in the registry context

Solves: Algorithm A "solves" Problem P.
Implements: Implementation I "implements" Algorithm A.
IsBasedOn: Realization R "is based on" Implementation I.
RunsOn: Implementation I "runs on" Machine M.

Associations are themselves registry objects and correspondingly stored in the registry with links to a source and a target registry object.

3.4 Classification of Entities

A classification scheme (or taxonomy) is a hierarchical tree of concepts that structures a particular knowledge area. The ability to classify an object (i.e., to link it to a concept in a classification scheme) is an important feature of a registry, because it facilitates the process of discovering the object. An object in the registry may be classified multiple times in one or in multiple schemes.

ebXML allows to submit new classification schemes into the registry such that registry objects may be classified in these schemes. We used this feature to import as an example the GAMS (Guide to Available Mathematical Software) classification scheme by translating the published XML format of this scheme into a format accepted by the registry [2]. Figure 3 shows the scheme viewed in the registry browser. MSDL entities may be classified in this and in other mathematical schemes.

3.5 Using the Entities

Since our mathematical entities are (extensions of) ebXML entities, they can be accessed using the standard ebXML mechanisms. For instance, Figure 4 demon-

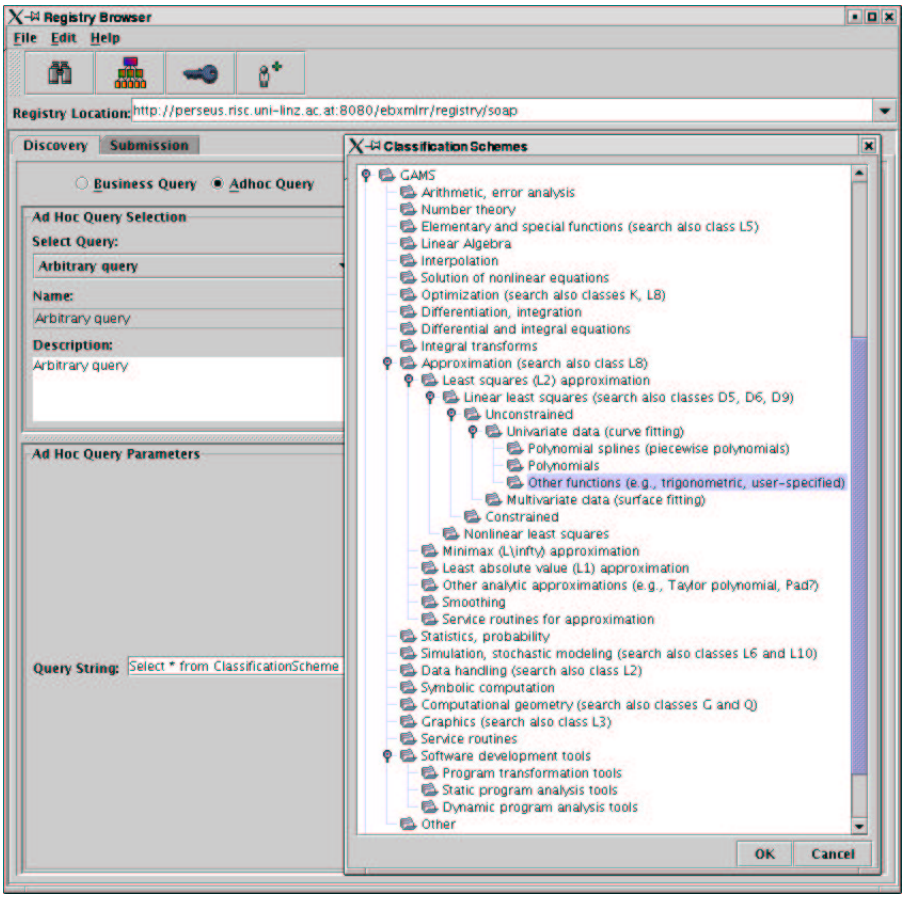

Fig. 3. Registry browser screenshot of GAMS classification scheme

strates how the ebXML browser displays sample entities (of type Service, Implementation, Problem, Algorithm, and Machine) with their names drawn in rectangular boxes and the relations among them illustrated as arrows.

3.6 Implementation Aspects

We implemented the entities of the service description model such that this implementation captures all the aspects and features of MSDL. Moreover the MSDL entity classes inherit all the functionality of the ebXML class "ExtrinsicObject" they extend. We also extended the management functionality of the ebXML registry to allow the registration, classification, association, and discovery of MSDL descriptions. The result of this implementation is a Java API for MSDL registries [13].

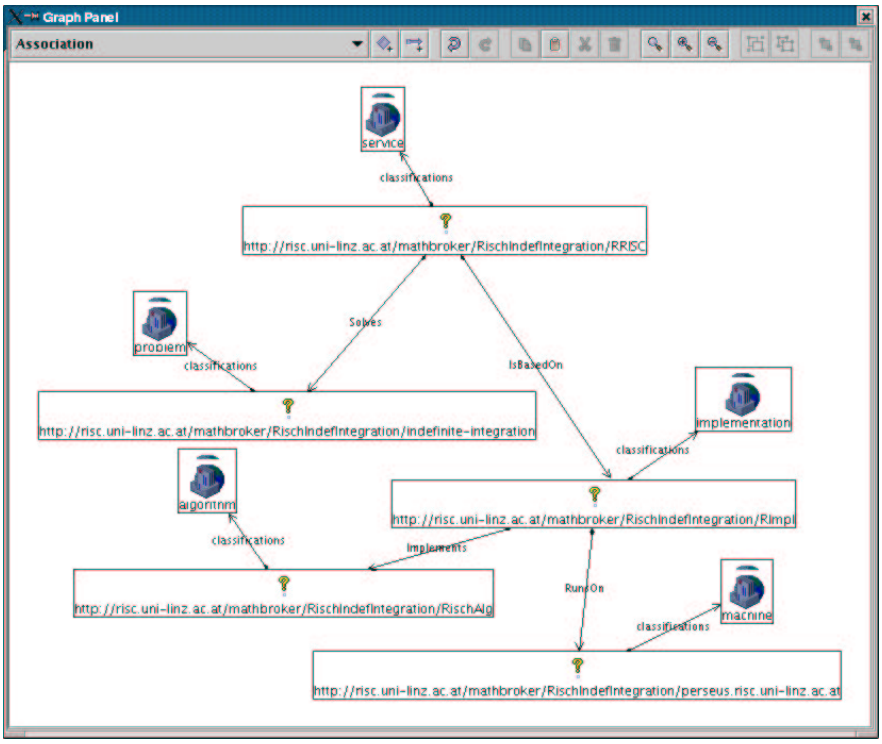

Fig. 4. Registry browser with MathBroker entities, their classifications and associations

4 Publishing and Querying Service Descriptions

We wrote a sample client that demonstrates the use of the registry API. The client performs two tasks: publishing, i.e., registering service descriptions, and querying, i.e., discovering them. For publishing, the client takes an MSDL file and registers all entities described in it (also creating the related associations and classifications). For querying, the client takes a question from the user and prints the resulting MSDL descriptions.

4.1 A Sample Description

A description can contain one or more service descriptions entities. A complete service description would include all the entities introduced in Section 2 and pointers for entity associations and classifications. A skeleton of such a description is shown below.

```
<\monet:definitions
  ...
```

```
  <mathb:machine_hardware name="perseus.risc.uni-linz.ac.at">
    ...
  </mathb:machine_hardware>

  <monet:problem name="indefinite-integration">
    ...
  </monet:problem>

  <monet:algorithm name="RischAlg">
    ...
  </monet:algorithm>

  <monet:implementation name="RImpl">
    ...
  </monet:implementation>

  <monet:service name="RRISC">
      ...
  </monet:service>
</monet:definitions>
```

4.2 Publishing to the Registry

The client, as shown below, makes a connection to the registry, uses this connection to obtain the registry service, utilizes the service by accessing the ebXML "life cycle manager" and "query manager" to publish to the registry.

The following part of the client code creates the connection to the registry; the essential information needed is the registry URL:

```
String url = "http://perseus:8080/ebxmlrr/registry/soap";
Properties props = new Properties();
props.setProperty("javax.xml.registry.queryManagerURL", url);
ConnectionFactory factory =
  MathBrokerConnectionFactoryImpl.newInstance();
factory.setProperties( props );
MathBrokerConnection connection =
    (MathBrokerConnection)factory.createConnection();
MathBrokerRegistryService mrs =
  connection.getMathBrokerRegistryService();
MathBrokerLifeCycleManager mlcm =
  mrs.getMathBrokerLifeCycleManager();
MathBrokerFocusedQueryManager fqm =
  mrs.getMathBrokerFocusedQueryManager();
```

The client takes an MSDL file (e.g. `risch.xml`) and extracts the description of each entity it contains. For each description it creates a registry ob-

ject embedding that description and also creates 0 updates all required associations and classifications. All this functionality is hidden in the method `publishMathBrokerObject` of the MSDL registry API:

```
File repositoryItemFile = new File ("risch.xml");
javax.activation.DataHandler repositoryItem =
  new DataHandler(new FileDataSource(repositoryItemFile));
mlcm.publishMathBrokerObject(fqm, repositoryItem);
```

4.3 Querying the Registry

The client allows to make queries for mathematical objects according to ID, name, or classification by invoking the following methods of the registry API.

```
executeQueryById(argument)
executeQueryByName(argument)
executeQueryByClassification(argument)
```

The implementation of the API ultimately invokes the MathBroker registry manager to retrieve a description from the repository, then extracts and displays the respective fields from these description.

The following example shows a query for an algorithm:

```
MathBrokerAlgorithm algorithm =
(MathBrokerAlgorithm)mlcm.createMathBrokerAlgorithm(
  null, null, classificationConcept, dh);
algorithm.showContent();
```

5 Conclusion

We presented first results on the development of a registry where the descriptions of mathematical services are published and can be discovered by potential clients. Our results demonstrate that standards and technologies that were originally developed for facilitating electronic business can be successfully used in a completely different (and considerably more sophisticated) application area, namely computer mathematics. Thus we profit from the work in the Web community, preserve compatibility with its standards, and build on its software.

This framework serves as the basis for our ultimate goal of developing a "semantic broker" where services register their problem solving capabilities, clients submit task descriptions, and a broker then determines the suitable services and returns them to the client for invocation. Our next steps will be the design of a more expressive query model based on the syntactic and semantic content of the registered descriptions and of a corresponding query language that allows clients to discover suitable services.

References

1. R. Ausbrooks et al. Mathematical Markup Language (MathML) Version 2.0. W3C Recommendation, October 2003. http://www.w3.org/TR/MathML2.
2. R. F. Boisvert, S.E. Howe, and D. K. Kahaner. GAMS: A Framework for the Management of Scientific Software, ACM Transactions on Mathematical Software 11(4), December 1985, 313–355.
3. O. Caprotti. Extending MONET to the MathBroker Information Model. Project Report, RISC-Linz, Johannes Kepler University, Linz, Austria, June 2003.
4. O. Caprotti, D. P. Carlisle, A. M. Cohen. The OpenMath Standard. The OpenMath Esprit Consortium. February 2000.
5. M. Dewar, D. Carlisle, O. Caprotti. Description Schemes For Mathematical Web Services. Proceedings of EuroWeb 2002 Conference: The Web and the GRID: from e-science to e-business. St Anne's College Oxford, UK, December2002 . British Computer Society Electronic Workshops in Computing (eWiC).
6. O. Caprotti and W. Schreiner. Towards a Mathematical Services Description Language. ICMS2002 , International Congress of Mathematical Software, Beijing, China, August 17-19, 2002.
7. ebXML Registry Information Model v2.0. OASIS, December 2001. http://www.oasis-open.org/committees/regrep/documents/2.0/specs/ebRIM.pdf
8. ebXML Registry Reference Implementation Project (ebxmlrr). OASIS, April 2004. http://ebxmlrr.sourceforge.net/
9. ebXML Registry Services Specification v2.0, OASIS, April 2002. http://www.oasis-open.org/committees/regrep/documents/2.0/specs/ebRS.pdf
10. Java API for XML Registries (JAXR). Sun Microsystems, April 2004. http://java.sun.com/xml/jaxr/
11. MapleNET. MapleSoft, April 2004. http://www.maplesoft.com/maplenet/
12. MathBroker: A Framework for Brokering Distributed Mathematical Services. Research Institute for Symbolic Computation, April 2004. http://poseidon.risc.uni-linz.ac.at:8080/mathbroker/index.xml
13. Mathbroker Registry API. Research Institute for Symbolic Computation, April 2004. http://poseidon.risc.uni-linz.ac.at:8080/results/registry/MBregistryAPI
14. Mathematical Service Description Language: Technical Report Deliverable D14, The MONET Consortium, March 2003. http://monet.nag.co.uk/
15. MathWeb.org: A Portal for Math-on-the-Web. University of Saarbrücken, April 2004, http://www.mathweb.org.
16. MONET: Mathematics on the Net. The MONET Consortium, April 2004. http://monet.nag.co.uk/
17. UDDI Version 2.04 API Specification. OASIS, July 2002. http://uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.pdf
18. WebMathematica, April 2004. http://www.wolfram.com/products/webmathematica/index.html
19. Web Services Activity. World Wide Web Consortium, March 2004. http://www.w3.org/2002/ws.