

Mathematical Service Matching Using Description Logic and OWL

Olga Caprotti¹, Mike Dewar² and Daniele Turi³

¹Dept of Mathematics
Technical University Of Eindhoven
Eindhoven
Netherlands
ocaprott@risc.uni-linz.ac.at

²NAG Ltd.
Wilkinson House
Jordan Hill Rd
Oxford OX2 8DR
United Kingdom
mike.dewar@nag.co.uk

³Information Management Group
Dept of Computer Science
University of Manchester
Oxford Road
Manchester M13 9PL
United Kingdom
dturi@cs.man.ac.uk

(Public)

Contents

1	Overview	2
2	General Design Issues and Rationale	2
3	The MONET Ontologies	3
3.1	GAMS	3
3.1.1	Symbolic	3
3.2	OpenMath	4
3.3	Hardware	4
3.4	Software	4
3.5	Problems	5
3.6	Algorithms	5
3.7	Directives	6
3.8	Theories	6
3.9	Bibliography	7
3.10	Encodings	7
3.11	MONET	8
4	From MSDL to OWL	8
5	Description Logics	8
6	Instance Store	11
6.1	Service Registration	12
6.2	Service Matching	12
7	Creating OWL Queries	12
8	Examples	16
9	Conclusions	18

1 Overview

The MONET ontologies are designed to model both queries and service descriptions: in that sense they are a formalism of the original MSDL language [13] which was deliberately designed to be “ontology neutral”. This was partly to make the task of authoring service descriptions more straightforward (since the language is quite compact) but mainly because the *Web Ontology Language (OWL)* [10] was then at an early stage of development. The ontologies are designed to be used by the broker and, in particular, the *Instance Store* [8] component which is responsible for matching queries to appropriate services. This component is, in turn, built on a Description Logic reasoner accessed through a generic interface. In our case we have chosen to use *RACER* [15] for this purpose, but there are several other alternatives which we could have used equally well.

The MONET ontologies were developed while OWL itself was still not finalised and, in consequence, we have had to work around numerous bugs and features of tools which were often in an early or incomplete state. This has effected the design and structure of the ontologies in a number of ways as described below. Despite these problems we are confident that the decision to be an “early adopter” of OWL was the right one as we expect it to be a key technology in the future semantic web. (On February 10, 2004 OWL has become the official W3C recommendation for web ontology languages.)

2 General Design Issues and Rationale

The MONET ontologies can be split into two broad classes: those which describe models developed within the project (for example those for software and hardware), and those which describe models defined by a third party (for example OpenMath and GAMS). To simplify development and also to reflect the fact that we are making use of third-party taxonomies (even though we have re-expressed them in OWL) we have structured the MONET ontology as a collection of individual ontologies which make use of each other via the owl `imports` statement¹.

Unfortunately when we started creating the ontologies the best editor available, Protégé [16], did not support owl:imports and so we had to design a tool to combine the separate ontologies into one. At the time of writing this is still necessary to classify the ontologies using RACER, or to use the ontologies with Instance Store.

Those parts of the ontology which model existing structures are, by their nature, fairly comprehensive. Those parts which we have introduced ourselves are, in general, less so. This is partly because they are only needed to demonstrate the effectiveness of our approach, but also because we believe that the ontologies should reflect real requirements rather than be designed in isolation.

Instance Store can be viewed as a collection of individuals described using some ontology. In the MONET case the individuals are service descriptions and the ontology is the one described in this document. The Instance Store accepts queries which are expressed as statements made using the ontology and returns all the individuals which match the query. A more technical description of the matching process is described in section 6.

Instance Store relies on the fact that its underlying ontology only contains classes, not instances. The result in the MONET ontology is that a number of objects which would most naturally be expressed as instances (for example OpenMath symbols, pieces of software etc.) have to be defined as classes and properties on them or relations between them described by restrictions on the properties rather than concrete values. On the other hand many mathematical concepts exist in a hierarchy even if they can be regarded as instances (mathematical problems and algorithms for example) and by insisting that everything is a class we can express this very naturally.

¹Unfortunately there are no existing ontologies in OWL which we were able to use. We had hoped to use the OWL version of Dublin Core at <http://www.aifb.uni-karlsruhe.de/WBS/rvo/ontologies> but it regards all the Dublin Core elements as annotations which is not suitable for our purposes.

Instance Store can only reason about objects in the ontology so we have had to create entries in the ontology for every concept and term that we want to use in service descriptions and queries. A corollary to this is that the MONET ontologies are not dynamically extensible — if a new concept is introduced into the ontology then Instance Store must be restarted with the new ontology.

There are a number of problems with concrete domains in RACER. These effectively limit us to using strings and integers as property values, even when types such as `xsd:anyURI` (in particular) would be better. Moreover RACER is overly sensitive about the contents of strings. They cannot contain a colon or look like a floating point number (so “1.0.1” is alright but “1.01” is not). This is why, for example, the version property of pieces of software tends to be expressed as e.g. “v1.1” rather than the more natural “1.1”.

Finally we note that the OWL abstract syntax parser in Instance Store has trouble parsing identifiers whose name has a span of characters matching a keyword (so for example if *domain* is a keyword then you cannot e.g. create a class in the ontology called *subdomainof*). Most of these problems have been eliminated in the software but there are still objects in the ontologies whose names reflect this (often through non-standard capitalisation).

3 The MONET Ontologies

As mentioned previously the MONET ontology is actually a collection of several individual “sub-ontologies”. We now describe each one in turn.

3.1 GAMS

The Guide to Available Mathematical Software (GAMS) [11] is a service offered by the National Institute for Science and Technology which provides an online index of available mathematical software classified according to the type of problems it solves. It is heavily biased towards numerical software (commercial packages from NAG, IMSL etc. and public-domain software such as LAPACK, LINPACK and the BLAS). Unfortunately, it is extremely poor at classifying other kinds of mathematical software package (symbolic, statistical, ...) because it classifies the package as a whole rather than individual modules within the package. Nevertheless as it is the only taxonomy of this type we decided to use it as a mechanism for describing services within the MONET framework.

The GAMS ontology is a simple class hierarchy which was automatically generated from NAG’s existing XML representation of the GAMS taxonomy. Each GAMS class corresponds to an OWL Class, and specialisation of a problem class is represented by a sub-class relationship. For example one-dimensional quadrature is a subclass of quadrature, and one-dimensional quadrature over a finite interval is a subclass of one-dimensional quadrature etc.

Objects in the GAMS ontology live in the GAMS namespace, for example the URI for quadrature is `http://gams.nist.gov#H2`.

3.1.1 Symbolic

To address GAMS’ shortcomings in the area of symbolic computation (an area of interest to several members of the Consortium) we have extended GAMS with a small taxonomy which is a sub-class of the GAMS “O” category (symbolic computation systems). Objects in this area are distinguished from true GAMS classes by being in the symbolic namespace, for example the URI for Real Algebraic Geometry is `http://monet.nag.co.uk/owl/symbolic#Real_Algebraic_Geometry`.

We could have introduced a completely new taxonomy in this case but decided against it for simplicity. As we shall describe later we have used sub-classing relationships between elements of the Problem ontology and GAMS to indicate that a particular problem is solved by software which conforms to a particular GAMS

class, and we would have had to introduce a new top-level class (e.g. `Software_Taxonomy`) to which all our taxonomy root classes belonged to preserve this scheme.

3.2 OpenMath

OpenMath [18] is a format for the representation of mathematical expressions and objects. Base terms in the language which have semantics attached to them are called symbols (for example *sin*, *integral*, *matrix*, ...) and groups of related symbols live in *content dictionaries*.

The OpenMath Ontology has one root class, `OpenMathSymbol`, whose subclasses correspond to the symbols defined in a particular content dictionary. The symbols themselves are defined as further subclasses of these: this is a case where instances would be more appropriate were it not for the restrictions imposed by Instance Store.

OpenMath is used in MONET for many purposes: to encode users' problems, to describe classes of problems, to describe the properties of parameters in service descriptions etc. Objects in the OpenMath ontology live in the OpenMath namespace, however because we wish to use the same mechanism as the OpenMath 2 standard² to construct URIs for OpenMath symbols we have different naming conventions for collections of symbols and for the individual symbols themselves. For example the symbol *sin* is defined in the content dictionary *Transcendental Functions 1*, usually abbreviated to *transc1*. In our ontology, the collection of symbols contained in *transc1* is the class `http://www.openmath.org/cd#transc1_Symbols` while the symbol *sin* itself is represented by the class `http://www.openmath.org/cd/transc1#sin`.

3.3 Hardware

The hardware ontology is designed to describe either classes of machines or individual machines. The idea is that a user might request that a service run on a particular model of machine (e.g. a Sun Enterprise 10000), or a general class of machine (e.g. shared memory), or a machine with a certain number of processors, or ... This will be used within the *Implementation* part of an MSDL service description.

There are two class hierarchies to describe the general type of hardware (the `Computer` class) and the `Manufacturer`. Computers are classified as serial or parallel, and in the latter case as shared or distributed memory. A particular model of computer, e.g. the Sun Enterprise 10000, is represented as a class whose parents are its manufacturer (in this case `Sun`) and its machine classification (in this case `SharedMemory`). A particular machine of this type may have properties indicating its actual model, operating system, and number of processors.

Objects in the hardware ontology live in the MONET namespace, for example the URI for the Sun Enterprise 10000 class in the ontology is `http://monet.nag.co.uk/owl#Enterprise10000`.

3.4 Software

The software ontology is designed to describe pieces of software. It is intended to be used in the implementation part of the service description and allows a user to express a preference for a service built using a particular piece of software in his or her query.

Within the ontology there are two class hierarchies. The first represents programming languages and the second software packages. Generally speaking software is classified into a general product line (for example the NAG C Library) whose sub-classes are particular versions (for example the NAG C Library mark 7). Software classes may have the properties `ImplementationLanguage`, `version` or `creator`. In the latter case we would have liked to use Dublin Core but the consensus seems to be that this is an ontology of annotations

²currently in draft form

which, as such, are ignored by the reasoner (so a user could not specify that the service selected to solve his or her problem must use software written by NAG Ltd. for example).

Objects in the software ontology live in the MONET namespace, for example the URI for the NAG C Library (mark 7) is http://monet.nag.co.uk/owl#NAG_C_Library_7.

3.5 Problems

This is designed to represent particular mathematical problems. MONET allows for a problem to be described in terms of its inputs and outputs, pre-conditions and post-conditions, using a specially written XML Schema [12].

Within the ontology each problem is represented as a class, which can have properties indicating bibliography entries and generalisations or specialisations of it. The most interesting property is `openmath_head` whose range is an object from the `OpenMathSymbol` class. This represents a particular symbol which can be used to construct an instance of the problem in question. So for example if a user sends the broker an integral of the form

$$\int_a^b f(x) dx$$

encoded in OpenMath the service matcher may infer that the problem which the user wants to solve corresponds to a sub-class of `Problem` whose `openmath_head` property has the value `calculus1_defint`. There is also an inverse property, `problem_kind`, which is not currently used.

As well as being sub-classes of `Problem`, particular problems may also belong to GAMS classes.

The inputs, outputs, pre- and post-conditions are not currently expressed in the ontology. To use the inputs and outputs in service matching would require at least their signatures to be represented in the ontology and this is quite hard, especially when one has to avoid instances. Simple signatures such as $\mathcal{R} \rightarrow \mathcal{R}$ are easy to encode but as soon as Tuples appear in the domain or range, e.g. $(\mathcal{R}, \mathcal{N}) \rightarrow \mathcal{R}$ then they become difficult to represent. Pre- and post-conditions could not be encoded in OWL in any reasonable way, although some of their properties could. This is an area which we would like to investigate further in future.

Objects in the problems ontology live in their own namespace, for example the URI for the simplest differentiation problem is <http://monet.nag.co.uk/problems/differentiation>. The reason why we do not use the MONET namespace here, or a fragment identifier to separate the last component of the URI, is because we wish to use URI-fragment identifiers to refer to the inputs/outputs etc. of the problem in the MSDL file. We have also arranged for the URI of the problem in the ontology to be the URL of the corresponding MPDL file.

3.6 Algorithms

The algorithms ontology <http://monet.nag.co.uk/algorithm#> is designed to serve as the backbone for the algorithm library in the MONET architecture. There are two subclasses in this ontology: `Algorithm`, which describes well-known algorithms for mathematical computations, and `Complexity`, which provides classes necessary for representing complexity information.

The `Algorithm` class has three subclasses, `Numeric`, `Symbolic`, and `SymbolicNumeric`, to represent computations that are approximate, exact and a mixture of the two. The subclasses of these are organized into problem areas. Figure 1 shows for instance the subclasses `Symbolic`, `Polynomial_System_Solving`, and `GroebnerBasis`, under which one finds <http://monet.nag.co.uk/algorithm#FGLM>. The property `bibliographic_reference` associates to an algorithm the bibliographical item for the resource which defines the algorithm. So for <http://monet.nag.co.uk/algorithm#FGLM>, the bibliographical item is the class <http://monet.nag.co.uk/bibliography#MR1263871> representing the paper MR1263871 on MathSciNet[3]. If the bibliographical item is not available in the bibliography ontology, then the reference can still be represented by using anonymous classes.

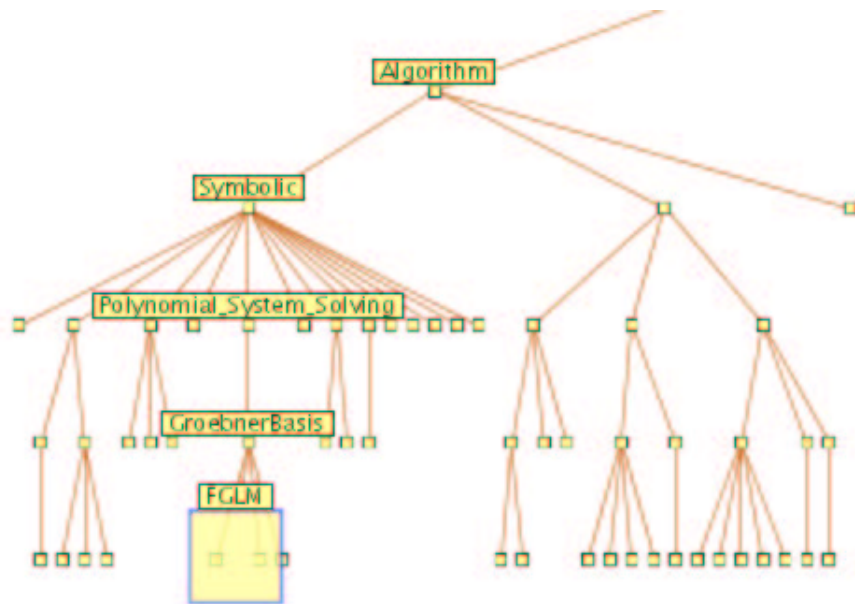


Figure 1: Direct subclasses of <http://monet.nag.co.uk/algorithm#> to FGLM

The `Complexity` class has three subclasses, `ComplexityClasses`, `ComputationModel` and `ResourceBound`. The `ComplexityClasses` hierarchy is divided according to the resource considered in `Space` and `Time`. For instance, <http://monet.nag.co.uk/algorithm#NP> is the class of languages recognizable by Nondeterministic Turing Machines in polynomial time. Its definition is expressed in the ontology by a restricted condition involving the properties `machine_model` and `time_bound` which relate to classes used to describe the theoretical computational models and the resource bounds occurring in the definitions of the complexity classes. For the definition of <http://monet.nag.co.uk/algorithm#NP>, the machine model is restricted to the class <http://monet.nag.co.uk/algorithm#NDTM>, identifying Nondeterministic Turing Machines, and the time bound is restricted to the class <http://monet.nag.co.uk/algorithm#Polynomial>, expressing the bound on a resource R “there is a constant k such that the resource R is bounded by $O(n^k)$, where n is the size of the input”.

3.7 Directives

The directives ontology is a collection of classes which identify the task that is performed by the service as described in [12, 13] – for example the directive `prove` is referred to as <http://monet.nag.co.uk/owl#prove>. Certain directives require a logical theory as content. This fact is mirrored in the ontology by the property `inTheory` which associates a class in the theory ontology described below to the directives http://monet.nag.co.uk/owl#prove_in_theory and http://monet.nag.co.uk/owl#decide_in_theory.

3.8 Theories

The theory ontology collects classes that represents available formalized theories in digital libraries of mathematics. In the current version, OMDoc theories [4] in the collections `logics` and `tps` are available as classes with own identifier in this ontology. Figure 2 displays this hierarchy.

A sibling class to the `OMDOC_theory` class is the `OpenMath_theory` class and can be populated in the future by theories formalized in OpenMath. Theories are described by their encoding and the source URL, respectively given by the properties `hasEncoding`, and `sourceURL`.

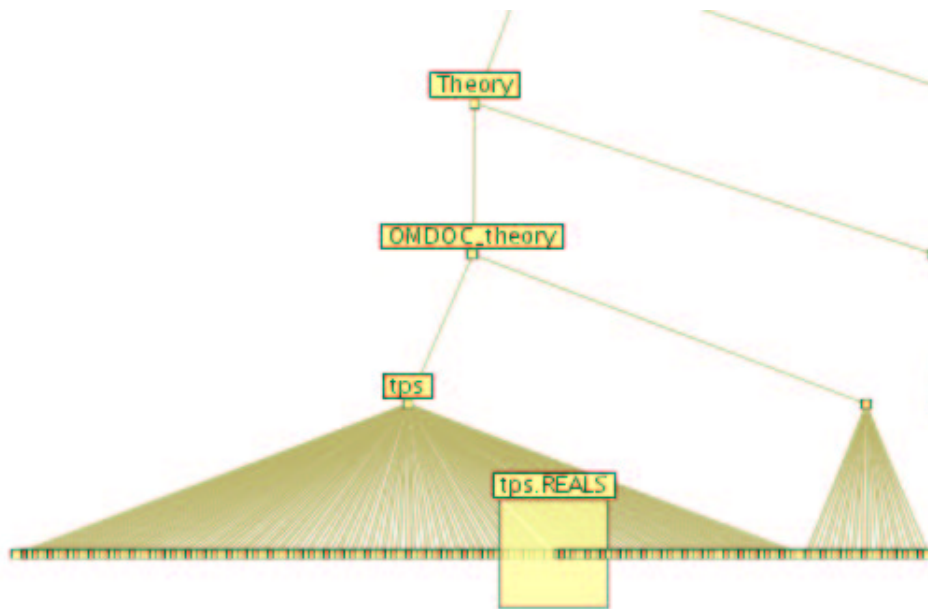


Figure 2: The class representing the OMDOC theory `tps.REAL`.

3.9 Bibliography

The ontology used to represent bibliographical data, <http://monet.nag.co.uk/bibliography#>, provides classes representing bibliography references. The collections of mathematical reviews available online are represented by subclasses of the class `Bibliography_Index`. Among them are `ZentralBlatt_MATH` and `MathSciNet`, respectively the class collecting reviews published in the ZentralBlatt MATH database [6] and in MathSciNet [3]. Bibliography databases specialized in mathematics, such as `CompuScience` [1] and `ERAM` [2] are subclasses of `Bibliography_DB`.

The datatype property `bibref` takes as value the unique code used in the respective systems to identify an item, e.g. 0805.13007 or MR1263871, are used to create classes which represent bibliographical resources. This ontology could be populated by classes representing the bibliographical references so that in order to cite a certain paper it would be enough to use the id of the corresponding class in the ontology. However, in the algorithm or problem ontology, it is also feasible to represent bibliography references by anonymous classes with the proper datatype restriction.

3.10 Encodings

The encodings ontology is a simple collection of classes which represent the formats used for encoding mathematical objects in the MONET framework — for example the formats read and written by the services or used to encode explanations. The URIs for the objects in this ontology are as far as possible the namespace identifiers (in the case of an XML format) or some other natural identifier. For example the URI for OpenMath is <http://www.openmath.org/OpenMath>.

Thus it is possible for a user to stipulate that they would like to use a service which takes OpenMath as input but returns MathML. These identifiers for particular encodings are also used in providing *explanations* [14].

3.11 MONET

The MONET ontology imports all the other ontologies and is used to represent complete service descriptions and queries. Not surprisingly the structure reflects that of MSDL [13] with an instance of the **Service** class having an associated **Classification** and **Implementation** class.

Neither service descriptions nor queries form part of the static ontology loaded by the Instance Store, but the OWL versions are created by the client manager and service registry components of the broker. The queries are represented as fragments of a service description, i.e. a “blueprint” for the kind of service being sought.

The possible properties which a service may have are as follows.

Classification:

Property	Class	Description
<code>service_problem</code>	Problem	The problem the service solves
<code>service_gams</code>	Gams	The GAMS classification of the service
<code>service_directive</code>	Directive	The directive the service conforms to
<code>input_format</code>	Encoding	The format for problem input
<code>output_format</code>	Encoding	The format for problem output

Implementation:

Property	Class	Description
<code>service_algorithm</code>	Algorithm	The algorithm used by the service
<code>service_software</code>	Software	Software used to implement the service
<code>service_platform</code>	Hardware	The machine platform on which the service runs
<code>service_algorithmic_property</code>	OpenMathSymbol	Extra information accepted or returned by the service

4 From MSDL to OWL

The structure of an MSDL file allows it to be transformed very naturally into OWL using the ontologies described above. We use the OWL *Abstract Syntax* [9] to express the service descriptions, because XML-RDFS representations are not suitable for expressing fragments of ontologies as required by descriptions. For this reason we developed an OWL abstract syntax parser that it is now part of the OWL API [5].

For instance, consider the simplified version of an MSDL document in Figure 3. We can apply an XSLT [19] stylesheet and obtain from it the description in OWL abstract syntax depicted in Figure 4, where, for simplicity, we have omitted some of the “`someValuesFrom(owl:Thing)`” restrictions needed to ensure the existence of some value for each property. Thus

```
restriction(property someValuesFrom(owl:Thing))
```

is needed not only for `service_classification` but also for every other property in the description.

5 Description Logics

OWL has a formal semantics as a *Description Logic (DL)*. Description logics [7] are *decidable* fragments of classic predicate logic (*FOL*). They include boolean combinators (including negation), unary and binary

```

<service name="nagopt">
  <classification>
    <gams_class>GamsG1a1a</gams_class>
    <problem>constrained_minimisation</problem>
    <input_format>OpenMath</input_format>
    <output_format>OpenMath</output_format>
    <directive>find</directive>
  </classification>
  <implementation>
    <software>NAG_C_Library_7</software>
    <platform>PentiumSystem</platform>
    <algorithm>
      Safeguarded_Quadratic-Interpolation
    </algorithm>
  </implementation>
</service>

```

Figure 3: Pseudo-MSDL for describing the service `nagopt`

predicate symbols, universal and existential quantification on binary predicates, base types (strings, integers, etc...), cardinality restrictions, and more. Computationally, the interest in DLs is that highly optimised reasoners (such as Fact, RACER, and now FaCT++) have been developed for them. This means that we can reason over OWL ontologies in an *effective* and decidable way, and with classic FOL semantics.

We illustrate the semantics of OWL by showing the FOL translation of the description in Figure 4:

$$\begin{aligned}
D(x) &\equiv \text{Service}(x) \\
&\wedge \forall y. (\text{service_classification}(x, y) \Rightarrow C(y)) \\
&\wedge \forall y. (\text{service_implementation}(x, y) \Rightarrow I(y))
\end{aligned}$$

where

$$\begin{aligned}
C(y) &\equiv \text{Classification}(y) \\
&\wedge \forall z. (\text{gams_class}(y, z) \Rightarrow \text{GamsG1a1a}(z)) \\
&\wedge \forall z. (\text{service_problem}(y, z) \Rightarrow \text{constrained_minimization}(z)) \\
&\wedge \forall z. (\text{input_format}(y, z) \Rightarrow \text{OpenMath}(z)) \\
&\wedge \forall z. (\text{output_format}(y, z) \Rightarrow \text{OpenMath}(z)) \\
&\wedge \forall z. (\text{directive}(y, z) \Rightarrow \text{find}(z))
\end{aligned}$$

and

$$\begin{aligned}
I(y) &\equiv \text{Implementation}(y) \\
&\wedge \forall z. (\text{software}(y, z) \Rightarrow \text{NAG_C_Library_7}(z)) \\
&\wedge \forall z. (\text{platform}(y, z) \Rightarrow \text{PentiumSystem}(z)) \\
&\wedge \forall z. (\text{algorithm}(y, z) \Rightarrow \text{Safeguarded_QuadraticInterpolation}(z))
\end{aligned}$$

OWL Individuals are interpreted as FOL constant terms. Therefore, semantically, registering a service s amounts to stating that $D(s)$ holds. The fact that the MONET ontologies have no individuals means that they can be interpreted as a set \mathcal{T} of open FOL formulas with no constants. We have then a neat distinction between the open formulas in \mathcal{T} and the ground formulas in \mathcal{A} — the set of assertions about the individuals. In terms of FOL deduction relation ‘ \vdash ’, this means that querying for the services matching a description Q

```

intersectionOf(
  <http://monet.nag.co.uk/owl#Service>
  restriction(
    <http://monet.nag.co.uk/owl#service_classification>
    someValuesFrom(owl:Thing))
  restriction(
    <http://monet.nag.co.uk/owl#service_classification>
    allValuesFrom(
      intersectionOf(
        <http://monet.nag.co.uk/owl#Classification>
        restriction(
          <http://monet.nag.co.uk/owl#gams_class>
          allValuesFrom(<http://gams.nist.gov#GamsG1a1a>))
        restriction(
          <http://monet.nag.co.uk/owl#service_problem>
          allValuesFrom(<http://monet.nag.co.uk/problems/constrained_minimisation>))
        restriction(
          <http://monet.nag.co.uk/owl#input_format>
          allValuesFrom(<http://www.openmath.org/OpenMath>))
        restriction(
          <http://monet.nag.co.uk/owl#output_format>
          allValuesFrom(<http://www.openmath.org/OpenMath>))
        restriction(
          <http://monet.nag.co.uk/owl#service_directive>
          allValuesFrom(<http://monet.nag.co.uk/owl#find>))
      )
    )
  )
  restriction(
    <http://monet.nag.co.uk/owl#service_implementation>
    someValuesFrom(owl:Thing))
  restriction(
    <http://monet.nag.co.uk/owl#service_implementation>
    allValuesFrom(
      intersectionOf(
        <http://monet.nag.co.uk/owl#Implementation>
        restriction(
          <http://monet.nag.co.uk/owl#service_software>
          allValuesFrom(<http://monet.nag.co.uk/owl#NAG_C_Library_7>))
          restriction(
            <http://monet.nag.co.uk/owl#service_platform>
            allValuesFrom(<http://monet.nag.co.uk/owl#PentiumSystem>))
          restriction(
            <http://monet.nag.co.uk/owl#service_algorithm>
            allValuesFrom(
              <http://monet.nag.co.uk/algorithm#Safeguarded_Quadratic-Interpolation>))
        )
      )
    )
  )
)

```

Figure 4: Description of service nagopt in OWL abstract syntax

amounts to retrieving the set

$$\{s \mid \mathcal{T} \vdash D(x) \Rightarrow Q(x) \text{ for some } D(s) \in \mathcal{A}\}$$

In DL terminology, the formula ‘ $D(X) \Rightarrow Q(x)$ ’ is read as ‘ D is subsumed by Q ’ and the above is the set of services which are instances of Q (with respect to the ontology \mathcal{T} and the assertions \mathcal{A}).

6 Instance Store

We have mentioned that DL reasoners are effective and highly optimised. Yet they have scalability problems which make them best suited for reasoning about the relatively limited knowledge in the ontology \mathcal{T} (thousands or tens of thousands of classes) rather than the assertions in \mathcal{A} which might easily reach millions of ground facts.

To tackle this problem we have developed a Java component called *Instance Store (IS)* [20]. The IS stores the assertions in a database, together with their descriptions and additional information inferred from a DL reasoner. Queries to the instance store are efficiently answered by using a combination of DL reasoning and database queries. Most importantly, very large numbers of assertions can be handled, while still providing sound and complete answers for arbitrary query terms. All this is based on the assumption that the ontology itself does not contain individuals and that the assertions themselves are *role-free* in the sense that no relation between individuals is asserted, which is the case for service registration/matching.

Figure 5 depicts the IS architecture, while Figure 6 depicts the logical functionalities performed by IS of service registration and matching within the MONET broker architecture.

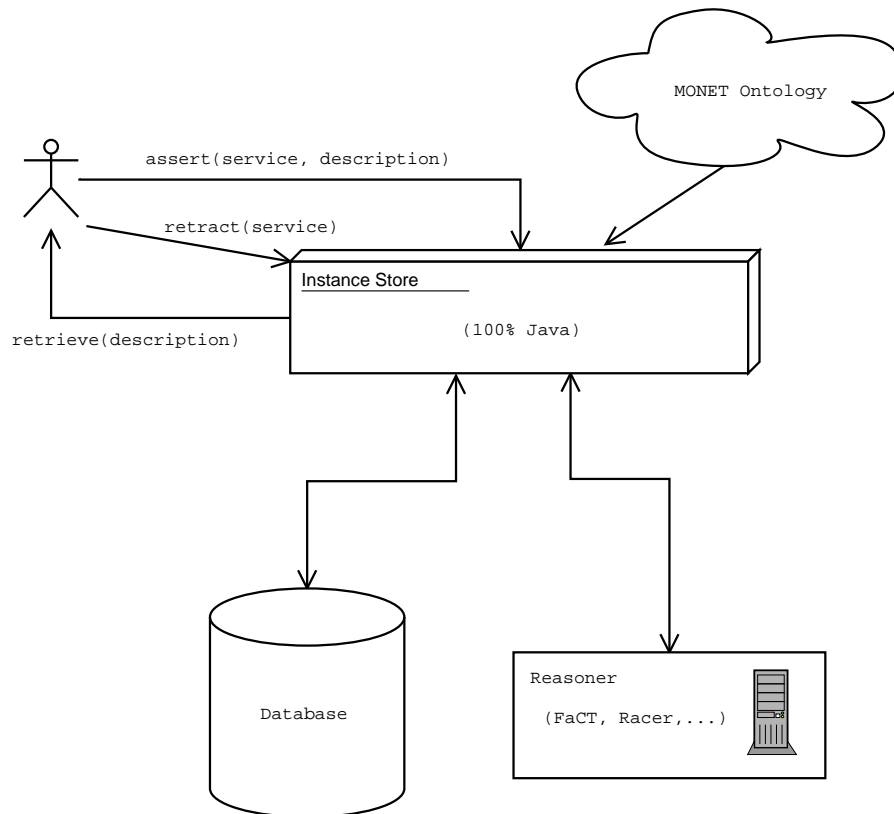


Figure 5: IS Architecture

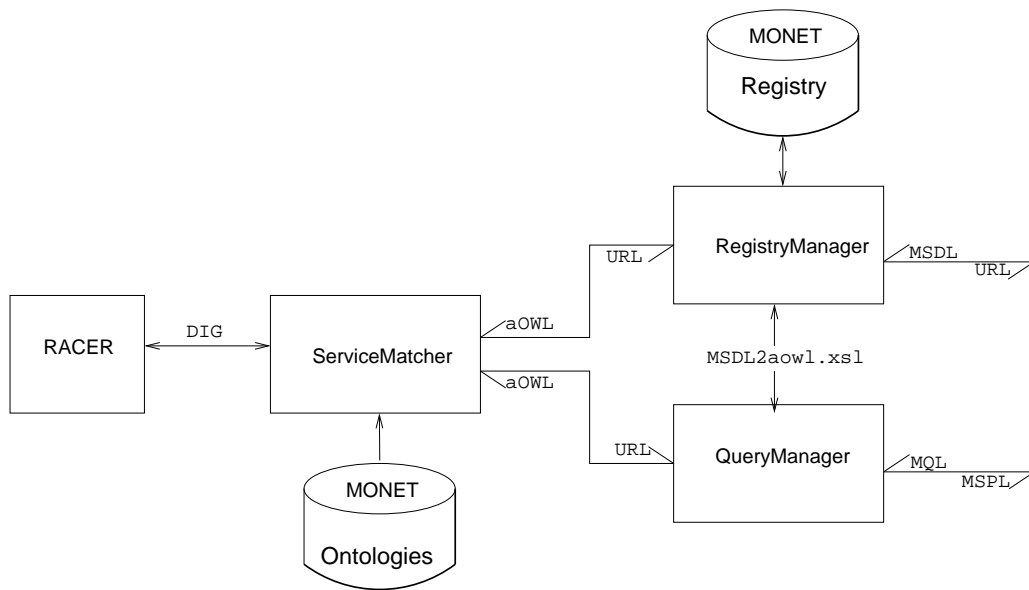


Figure 6: MONET Architecture of Service Manager and Query Manager

6.1 Service Registration

The process of registering a service consists of submitting the service URI together with its OWL description to the IS. The IS stores all such pairs in a database (the MONET Registry in Figure 6), but also infers with the reasoner all the concepts in the MONET ontology which are more general than or equivalent to the given description, associating them with the service URI in the database as well. This information will be used to match queries.

6.2 Service Matching

Service matching is performed by submitting a retrieval query to the IS. The query itself consists of an OWL description of the required service. The IS will answer by looking for all the services (URIs) in its database which are instances of the query description.

In order to retrieve all instances of a description Q , the IS first uses the reasoner to find the concepts in the ontology which are subsumed by Q and then retrieves the corresponding services in the database.

Next, the IS asks the reasoner whether Q is equivalent to a concept C in the ontology. If that is the case then also the services corresponding to C in the database are returned and the procedure stops.

If, however, the description is not equivalent to any concept in the ontology, the IS needs to retrieve all the services which are instances of immediate parents of Q and check (again through the reasoner) whether those services are, in fact, instances of the given description. If they are, then the relevant services are also returned. (For full details see [8].)

7 Creating OWL Queries

When a service registers itself with the broker it submits a service description in MSDL. The registry manager component of the broker translates this into OWL using an XSLT stylesheet (`MSDL2aowl.xsl` in Figure 6).

The simplest queries are just fragments of MSDL which provide a template for the service the user wishes

to use. These queries are translated into OWL by the query manager component of the broker using XSLT (MSDL2aowl.xsl in Figure 6). A simple example would consist of just a *classification* which contains a reference to a known problem (i.e. one contained in the ontology) along with values for the input data. In this case, in effect, the user is stating that he or she would like to solve this instantiation of this particular problem. Using this approach one may constrain the software or hardware the service runs on, the algorithm to be used or anything else which can be expressed in MSDL. Note that the class hierarchy in the ontology plays an important part here: given the following fragment of the algorithm ontology:

- Algorithm
 - ...
 - Quadrature
 - Automatic_Adaptive_Methods
 - ...
 - Automatic_Non-Adaptive_Methods
 - ...
 - Fixed_Abscissae_Methods
 - Gauss-Hermite
 - Gauss-Laguerre
 - Gauss-Legendre
 - Gauss-Rational
 - Gill-Miller
 - ...

a user could request an algorithm which used fixed abscissae and be satisfied with any one of the five alternatives.

Queries are not, however, restricted to simple templates of the service the user wants. The query manager can also construct OWL specifications based on other user inputs, such as that given in figure 7. In this case the user has said that he or she would like the service to return the result of computing a particular integral, namely:

$$\int_{0.0}^{1.0} \sin(x) dx$$

This is a very natural way to express this question from a mathematical point of view. Since there is no explicit problem, taxonomy or algorithm element to help the service matcher, it is clear to a human reader³ that this is a definite integration problem. OpenMath has a tree structure and the root node for an OpenMath object can only be one of a small number of elements, in this case an *OMA* or *application* element. Since we know the structure of such elements we know that if its first child is an *OMS* or *OpenMath Symbol* then this is in effect the principle constructor for the object. In this case the symbol is that one used to represent definite integration (**defint** from the content dictionary **calculus1**).

Elements of the Problem Ontology may have the property **openmath_head** which indicates an OpenMath symbol which can be used to construct instances of that problem. In fact the **definite_integration** class in the Problem ontology has an **openmath_head** property of **defint**. So if the Query Manager transforms this query to a request to solve problems with an **openmath_head** property of **defint** then the reasoner will infer that any service solving the **definite_integration** problem is a candidate.

```

<monet:query xmlns:monet="http://monet.nag.co.uk/monet/ns"
             xmlns:om="http://www.openmath.org/OpenMath">
  <monet:problem >
    <monet:header/>
    <monet:body>

      <monet:output name="integral">
        <monet:value>
          <om:OMOBJ>
            <om:OMA>
              <om:OMS name="defint" cd="calculus1"/>
              <om:OMA>
                <om:OMS name="interval" cd="interval1"/>
                <om:OMF dec="0.0"/>
                <om:OMF dec="1.0"/>
              </om:OMA>
              <om:OMBIND>
                <om:OMS name="lambda" cd="fns1"/>
                <om:OMBVAR>
                  <om:OMV name="x"/>
                </om:OMBVAR>
                <om:OMA>
                  <om:OMS name="sin" cd="transc1"/>
                  <om:OMV name="x"/>
                </om:OMA>
              </om:OMBIND>
            </om:OMA>
          </om:OMOBJ>
        </monet:value>
      </monet:output>

    </monet:body>
  </monet:problem>
</monet:query>

```

Figure 7: A Query Involving a natural formalisation of a Mathematical Problem

```

<service name="nagquad">
  <classification>
    <gams_class>GamsH2a1a1</gams_class>
    <problem>definite_integration</problem>
    <input_format>OpenMath</input_format>
    <output_format>OpenMath</output_format>
    <directive>find</directive>
  </classification>
  <implementation>
    <software>NAG_C_Library_7</software>
    <platform>PentiumSystem</platform>
    <algorithm>de_Doncker</algorithm>
  </implementation>
</service>

```

Figure 8: Pseudo-MSDL for nagquad

```

<service name="nagroot">
  <classification>
    <gams_class>GamsF2</gams_class>
    <problem>zero_of_nonlinear_system</problem>
    <input_format>OpenMath</input_format>
    <output_format>OpenMath</output_format>
    <directive>find</directive>
  </classification>
  <implementation>
    <software>NAG_C_Library_7</software>
    <platform>PentiumSystem</platform>
    <algorithm>Powell</algorithm>
  </implementation>
</service>

```

Figure 9: Pseudo-MSDL for nagroot

```

<service name="nagopt-variation">
  <classification>
    <gams_class>GamsG1a</gams_class>
    <problem>constrained_minimisation</problem>
    <input_format>algstr1.cdg</input_format>
    <output_format>OpenMath</output_format>
    <directive>find</directive>
  </classification>
  <implementation>
    <software>NAG_C_Library_7</software>
    <platform>SR8000</platform>
    <algorithm>Quadratic_Programming</algorithm>
  </implementation>
</service>

```

Figure 10: Pseudo-MSDL for nagopt-variation

8 Examples

We now illustrate service matching by considering five queries to an elementary IS consisting of four services. The four services are described by the pseudo-MSDL in Figures 3, 8, 9, and 10.

The first query, in Figure 11, asks for all individuals whose GAMS classification is **GamsG**. The IS 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>)  
    )  
  )  
)
```

Figure 11: Query for services with GAMS classification **GamsG**

The second query, in Figure 12, asks for all individuals whose implementation has **Root_Finding** as algorithm. In the **Algorithm** ontology we can find that **Powell** is a child of **Root_Finding**. And, indeed, the IS answer to the query is **nagroot**.

```
restriction(  
  <http://monet.nag.co.uk/owl#service_implementation>  
  someValuesFrom(  
    restriction(  
      <http://monet.nag.co.uk/owl#service_algorithm>  
      someValuesFrom(<http://monet.nag.co.uk/algorithm#Root_Finding>)  
    )  
  )  
)
```

Figure 12: Query for services with **Root_Finding** algorithm

We can generalise the query by replacing **Root_Finding** with **Numeric** as in Figure 13 and obtain all services as answers, since they are all numeric.

A more interesting query is in Figure 14, where we ask for all individuals with algorithm having a **Zentral Blatt** bibliographic reference *0277.65028*. The answer is again **nagroot**, since in the **Bibliography** ontology we have associated that reference to **Powell**.

We now illustrate the use of negation in a query. Consider first the query in Figure 15 where we ask for all individuals running on a platform whose number of processor is one. The answer is **nagopt**, **nagroot**, and **nagquad**, but not **nagopt-variation** since we asserted it runs on *SR8000* which is a parallel processor. Now, if we add to the **Hardware** ontology the axiom that **Serial** and **Parallel** are disjoint, we can infer that not **Parallel** is **Serial**, hence the query in Figure 16 would return again **nagopt**, **nagroot**, and **nagquad**.

³who is conversant with OpenMath

```

restriction(
  <http://monet.nag.co.uk/owl#service_implementation>
  someValuesFrom(
    restriction(
      <http://monet.nag.co.uk/owl#service_algorithm>
      someValuesFrom(<http://monet.nag.co.uk/algorithm#Numeric>)
    )
  )
)

```

Figure 13: Query for services with Numeric algorithm

```

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>)
              )
            )
          )
        )
      )
    )
  )
)

```

Figure 14: Query for services with algorithm with a Zentral Blatt reference *0277.65028*

```

restriction(
  <http://monet.nag.co.uk/owl#service_implementation>
  someValuesFrom(
    restriction(<http://monet.nag.co.uk/owl#service_platform>
      someValuesFrom(
        restriction(
          <http://monet.nag.co.uk/owl#number_of_processors>
          value("1"^^<http://www.w3.org/2001/XMLSchema#int>)
        )
      )
    )
  )
)

```

Figure 15: Query for services running on a single processor platform

the individual functions available within a comprehensive software package such as the NAG Library or Maple.

References

- [1] Computer Science Database. <http://www.zblmath.fiz-karlsruhe.de/COMP/quick.html>.
- [2] Electronic Research Archive for Mathematics (ERAM). <http://www.emis.de/projects/JFM/>.
- [3] MathSciNet, Mathematical Reviews on the Web. <http://www.ams.org/mathscinet>.
- [4] OMDoc: A Standard for Open Mathematical Documents. <http://www.mathweb.org/omdoc/>.
- [5] OWL API. <http://sourceforge.net/projects/owlapi>.
- [6] Zentralblatt math. <http://www.emis.de/ZMATH/>.
- [7] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider, editors. *The Description Logic Handbook — Theory, Implementation and Applications*. Cambridge University Press, 2003.
- [8] S. Bechhofer, I. Horrocks, and D. Turi. Instance store - database support for reasoning over individuals. <http://instancestore.man.ac.uk/instancestore.pdf>, 2002.
- [9] Sean Bechhofer, Peter F. Patel-Schneider, and Daniele Turi. OWL Web Ontology Language Concrete Abstract Syntax. Technical report, The University of Manchester, December 2003. Available from <http://owl.man.ac.uk/2003/concrete/latest/>.
- [10] Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. OWL Web Ontology Language Reference. Technical Report REC-owl-ref-20040210, The Worldwide Web Consortium, February 2004. Available from <http://www.w3.org/TR/2004/REC-owl-ref-20040210/>.
- [11] 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):313–355, December 1985.
- [12] Olga Caprotti, David Carlisle, Arjeh Cohen, and Mike Dewar. The Mathematical Problem Ontology: final version. Technical Report Deliverable D11, The MONET Consortium, March 2003. Available from <http://monet.nag.co.uk>.
- [13] The MONET Consortium. Mathematical Service Description Language: Final version. Technical Report Deliverable D14, The MONET Consortium, March 2003. Available from <http://monet.nag.co.uk>.
- [14] James Davenport. The Mathematical Explanation Ontology: draft. Technical Report Deliverable D07, The MONET Consortium, March 2003. Available from <http://monet.nag.co.uk>.
- [15] Volker Haarslev and Ralf Moller. Description of the RACER system and its applications. In Rajeev Gore, Alexander Leitsch, and Tobias Nipkow, editors, *Automated reasoning: First International Joint Conference, IJCAR 2001, Siena, Italy, June 18–23, 2001: proceedings*, volume 2083 of *Lecture Notes in Artificial Intelligence*, New York, NY, USA, 2001. Springer-Verlag Inc.
- [16] N.F. Noy, M. Sintek, S. Decker, M. Crubezy, R.W. Fergerson, and M.A. Musen. Creating semantic web contents with protege-2000. *IEEE Intelligent Systems*, 16(2):60–71, 2001.
- [17] D. Richardson. Some Unsolvable Problems Involving Elementary Functions of a Real Variable. *Journal of Computational Logic*, 33:514–520, 1968.

- [18] The OpenMath Society. The OpenMath Standard, October 2002. Available from <http://www.openmath.org/standard/om11/omstd11.xml>.
- [19] XSL Transformations (XSLT) Version 1.0. Technical Report REC-xslt-19991116, The Worldwide Web Consortium, November 1999. Available from <http://www.w3.org/TR/1999/REC-xslt-19991116>.
- [20] D. Turi. Instance Store. <http://instancestore.man.ac.uk>.