TOWARDS A MATHEMATICAL SERVICES DESCRIPTION LANGUAGE

OLGA CAPROTTI
WOLFGANG SCHREINER
Research Institute for Symbolic Computation
Johannes Kepler University, A-4040 Linz, Austria
{ocaprott,schreine}@risc.uni-linz.ac.at

As a foundation for a framework for brokering mathematical services distributed among networked servers, we propose a language that describes the mathematical problems solved by the services. This mathematical service description language extends WSDL, the web service description language. Servers register their problem solving capabilities expressed in this language with a "semantic broker" to which clients submit corresponding task descriptions. The broker (possibly in cooperation with a deduction system) determines the suitable services and returns them to the client for invocation. This mechanism thus hides from the client the actual implementation of mathematical services and focuses on the semantical aspects.

1 Introduction

In the recent years, mathematical software systems, i.e., computer algebra systems like Maple, Mathematica, or GAP, automated theorem provers like Otter, PVS, or Theorema, or systems for numerical scientific computing like Matlab are slowly migrating from being standalone desktop tools to be used by a human to computational engines used as "mathematical services" by other programs. There are numerous examples in the literature of scientific software needing access to computer algebra methods, of computer algebra systems requiring advanced reasoning capabilities vice-versa of automated theorem provers exploiting computer algebra systems for their system solving facilities, see e.g. [1]. Some systems provide built-in communication protocols for interfacing to third-party software, whereas others have to be accessed via the standard input/standard output interface. Frequently however these approaches suffer from the inadequate specification of the functionality of a system. The result may take surprising forms, a challenge for the interpretation capabilities of the client software sometimes forced to mimic a human user.

Furthermore the Internet and the World Wide Web have further extended the range of possibilities by enabling networked access to mathematical services hosted in distant locations. Clients may use these services via remote access protocols [2,3,4]. The more networked mathematical services become available, the less it is desirable that each speaks its own language, therefore common mathematical communication protocols are required. Since more and more clients are programs rather than humans, mechanisms to convey some understanding on the meaning of the offered services need to be worked out.

The technologies for accessing mathematical services over the network focus mainly on the problem of portable data representation (OpenMath [5], MathML [6], MathBus [7], IAMC [2], MathWeb [8], JavaMath [3]). IAMC does provide a protocol for the exchange of meta-data about the names of services offered and about accepted data formats, but it does not yet explicitly address the problem how a client can learn about the service semantics. Only the MathWeb activities deal with the semantic level focusing on the integration of automated theorem provers.

Thus, while the basic communication protocols for mathematical data have emerged, there has been only little work on the higher protocol levels that make mathematical services actually usable for automated tools without human insight. In particular, additional protocol levels have to be designed in order to support programming of mathematical applications that employ mathematical services distributed over the Internet. The final goal of a mathematical service description language is to be able to write such programs from an abstract point of view, without knowing about the actually available implementations. In this paper we advocate a mathematical service description format that abstracts from the details of an implementation and focuses on the actual *problem* being solved. The paper begins with a section on some of the considerations that contributed to our design of a language for mathematical services. Section 3 explains how the Web Service Description Language can be extended by additional meta-data to become more useful for our purposes. The following section describes ongoing work on what we believe is a promising way of describing the implementation meta-data associated with a service. The paper ends by some final remarks.

2 Description of Mathematical Services

From a pragmatic point of view, the description of a mathematical service is intended to facilitate the discovery of a certain computational facility, either abstractly from a specific mathematical question or more informally by a name of an algorithm or via any other item that might identify a piece of software. Keeping this in mind, the overall design of the description language is structured in layers which proceed from an abstract view, namely the problem addressed by the mathematical service, all the way down to the hostname-port address of the service as depicted in Figure 1.

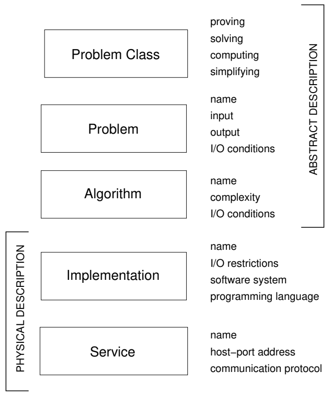

Figure 1. Mathematical Service Description Language Layers

2.1 Problem Class

We begin by attempting a classification/taxonomy of problem classes. This first coarse distinction is intended as a guideline on how to further specify a specific problem instance. The underlying idea is that a *computing* problem might have a different description from a *proving* problem. For instance, stating that a problem belongs to the class of *simplification* problems, amounts to choosing an equivalence relation $\equiv$ and a *reduction* relation since these are key components of the problem class definition.

A *problem class* description defines the structure of classes of problems (parametrized by a logical system L and a theory T of L) including for instance

- *Computing*, namely the problem of carrying out operations in a domain, in a theory T i.e., given a term f with a free variable x and a value a in a properly chosen domain, return the value of f with a substituted for x;
- *Simplifying*, namely the problem of obtaining simpler yet equivalent objects in a domain, i.e., given a term t and a set of simplification rules R, return a term $t' = t$ that satisfies some "simplicity" condition,
- *Solving*, i.e., given a set of formulas $F[\vec{x}]$ in a particular logic, find (some or all) substitutions $\vec{a}$ in a chosen domain that make $F[\vec{a}]$ true in a certain theory;
- *Proving*, i.e., given a closed formula F in a particular logic, determine whether the formula is true in a particular domain or not. In particular,

for some logic, this is equivalent to returning a proof.

In the logic broker architecture [9], a uniform way of specifying abstractly a service employs the notion of consequence relation [10]. Because this notion of consequence can be used for characterizing several known logics (including Linear Logic and non-monotonic logics) and in general, for semantics-independent classification of standard connectives (via equations on consequence relations), it is enough for formally specifying services provided by reasoning systems which are the focus of the logic broker architecture. Our approach aims at describing not only reasoning services (hence it must accommodate a description of consequence relations) but also computational ones.

2.2 Problem

A *problem* is a concrete instance of a problem class. The specific format of the problem instance is defined by the problem class it belongs to, eventually with additional (optional) fields that enhance its formalization. The name of a problem description may be used by a task to refer to a particular problem in a direct way; this allows a broker to identify corresponding services easily.

Additionally, if we consider that mathematical services as intended in this paper are computational services provided by mathematical software, it is possible to formally describe the problem that is solved as inspired by formal methods, that is using of techniques from logic and discrete mathematics supported by mechanized proof methods.

The motivation for including a formal definition is one of usage. The formal definition of a problem description can be used by a broker (possibly in collaboration with a deduction system) to identify services that satisfy a description given in terms of the content of a problem. A specification of a problem expresses constraints on what is to be computed in a style à la Floyd-Hoare by describing input and output parameters $\vec{x}$ and $\vec{y}$, their domains, and eventually a precondition $p(\vec{x})$ and a postcondition $q(\vec{x}, \vec{y})$. Notice that in order to formally define a problem, we will require a knowledge base of mathematics to use as reference for the mostly used concepts. Our preferred choice in this regard involves OpenMath.

For instance, a task requesting an integer GCD computation can be satisfied by the broker using a service advertising extended GCD computation. Although the names of the problems do not match, formal manipulation of the specification of the service allows to satisfy the request in an alternative way by reasoning on the preconditions and postconditions of the problem.

2.3 Algorithm

An *algorithm* is a solution strategy for a problem (possibly, by strengthening the precondition or weakening the postcondition, of a more special one). Apart from referring to the solved problem, an algorithm description may contain information about its asymptotic time complexity and memory requirements, i.e., about its general dynamic behavior. Potentially, also a description of the algorithm in a formal algorithmic language may be added for reasoning about additional properties. Apparently, there may be multiple algorithms for a given problem.

This layer captures the abstract computational essence of a mathematical service, which often is the interface of an algorithm. Requiring a certain algorithm for solving a problem is the natural mathematical activity it aims to support.

2.4 Implementation

An *implementation* is a particular realization of an algorithm as executable software, possibly with additional constraints on the input and additional possibilities for the output (error cases, implementation restrictions, etc). Apart from referring to the algorithm applied (or at least to the problem solved), an implementation description may contain additional information about the underlying software system, about its efficiency (relative to a fixed architecture and a fixed performance factor). Sometimes it is interesting to know the author of a certain implementation, and the date or version. This kind of meta-data is the object of this description layer.

2.5 Service

A *service* is a set of implementations running on a particular machine; its description contains information about the underlying machine (contact address, computational performance, communication performance, memory equipment, ...), about the actual communication protocol for invoking and controlling the service (including the choice of synchronous or asynchronous communication, the possibilities for querying the status of a computation, the potential of aborting a computation, etc), for querying the status of the machine, and about the way in which service requests are handled (one at a time, in parallel up to some limit, etc). This description is mostly achieved using WSDL [11].

In summary, every description has a unique resource identifier (URI) and may embed a formal definition. Less sophisticated applications may associate a fixed meaning with a URI; this is analogous to the standard set of OpenMath CDs which informally agree on the meaning of every symbol. More advanced applications may refer to the formal definition to learn about unknown entities; this is analogous to KQML-based approaches for describing mathematical services. While the first strategy makes the description framework quickly usable, the formal ultimately unleashes its power; combinations of both strategies (formal descriptions relying on informally defined entities) are also possible.

3 Service Layer: Exploiting WSDL

In order to have a running service it would suffice to limit the description to the service layer, and in fact most available client-server architectures work at this level and hide the location details and the objects' encoding to the client. This is the level of WSDL, the Web Services Description Language (WSDL) [11], currently only a W3C note, but already supported by several tools and programming interfaces for quickly developing services based on it.

A WSDL document is a collection of definitions aiming at specifying *services* as collections of *ports*, i.e. network endpoints. Several elements are used in the description of services: `<Types>` is a container for data types given according to a certain type system. In particular, WSDL supports the XML Schemas specification as preferred language for type definition but it also allows using other languages via extensibility. `<Message>` is an abstract, typed definition of the data being communicated in the input and output. `<Operation>` is an abstract description of an action supported by the service in terms of input and output. `<Port Type>` is an abstract set of operations supported by one or more endpoints. Here below is the abstract portion of a WSDL specification consisting of a service of type `Integrator` providing two operations. For reason of space, we have abbreviated some of the details.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
            targetNamespace="http://.../services/SymbolicIntegrator"
            xmlns="http://schemas.xmlsoap.org/wsdl/"
            xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
            xmlns:impl="http://.../services/SymbolicIntegrator-impl"
            xmlns:intf="http://.../services/SymbolicIntegrator"
            xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
            xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <import namespace="some-URI" location="some-URI"/>*
  <wsdl:documentation .... />?

  <wsdl:types>?
```

```
    <wsdl:documentation .... />? <-- extensibility element -->*
    <xsd:schema .... />*
  </wsdl:types>

  <wsdl:message name="DefIntegralResponse" .../>
       <wsdl:documentation .... />?
       <wsdl:part name="return" type="xsd:string"/>
       <wsdl:part name="nmtoken" element="qname"? type="qname"?/>*
  </wsdl:message>

  <wsdl:message name="DefIntegralRequest" .../>
  <wsdl:message name="nmtoken" .../>*

  <wsdl:portType name="Integrator">
    <wsdl:operation name="DefIntegral" parameterOrder="in0">
      <wsdl:input message="intf:DefIntegralRequest"/>
      <wsdl:output message="intf:DefIntegralResponse"/>
    </wsdl:operation>
    <wsdl:operation name="IndefIntegral" parameterOrder="in0" .../>
  </wsdl:portType>
```

The concrete set of definitions is given using two more elements. <Binding> is the association of a concrete protocol and data format specification to a particular abstract port type. For instance, the information that a service sends OpenMath data using the XML encoding of OpenMath is given in the binding. Finally, <Service> is a collection of related endpoints, (<Port>s), each defined as a combination of a binding and a network address. In order to know the functionalities of the service IntegratorService we have to check its portType, referenced by its binding. This describes the abstract syntactical interface in terms of operations with their input and output types.

```
  <wsdl:binding name="SymbolicIntegratorSoapBinding" type="intf:Integrator" .../>
       <wsdl:documentation .... />? <-- extensibility element -->*
       <wsdl:operation name="nmtoken">*
          <wsdl:documentation .... />? <-- extensibility element -->*
          <wsdl:input|output>?
              <wsdl:documentation .... />? <-- extensibility element -->
          </wsdl:input|output>
          <wsdl:fault name="nmtoken">*
              <wsdl:documentation .... />? <-- extensibility element -->*
          </wsdl:fault>
          </wsdl:operation>
    </wsdl:binding>

  <wsdl:service name="IntegratorService">
    <wsdl:port binding="intf:SymbolicIntegratorSoapBinding" name="SymbolicIntegrator">
      <wsdlsoap:address location="http://.../services/SymbolicIntegrator"/>
    </wsdl:port>
  </wsdl:service>

</wsdl:definitions>
```

Our experiments with WSDL[a] have shown that although it is very convenient to setup a service using WSDL, and to generate automatically clients for a service described by a WSDL specification, in order to reason about the service, more information is needed, for instance it would be desirable to know the abstract type of the arguments (that is, beyond the primitive types provided by XML schemas). Meta-data concerning the hardware type, its performance and details on the computational package used in the underlying implementation of the service is often useful for choosing a certain mathematical service instead of another one.

A key design feature of WSDL are the extensibility elements, used to include details on the transport protocol adopted by the service. Extensibility allows to modify the rule for the `<Binding>` in order to support a number of concrete communication protocols like SOAP, HTTP_GET_POST, SMTP_MIME. The extensibility fields of WSDL are the natural place where to convey additional meta-data of the kind we are interested in. We have decided to exploit extensibility within the `port` element in order to include in the WSDL meta-data concerning the hardware of the physical machine connected to the endpoint. This meta-data describes e.g. the processor type, the amount of memory, and a measure of performance using the Standard Performance Evaluation Corporation reference [b]. We have produced an XSD schema to express software and hardware characteristics to be included in a WSDL.

```
<wsdl:port binding="intf:SymbolicintegratorSoapBinding" name="SymbolicIntegrator">
  <wsdlsoap:address location="http://.../services/SymbolicIntegrator"/>
  <sdl:machine xmlns="http://perseus.risc.uni-linz.ac.at:8080/msdl/service">
     <processor>Asus A7V Athlon</processor>
     <memory>1.2GHz</memory>
     <performance SPECtype="SPEC CINT2000" metric="speed">
        <CPU value="1"/> <base value="409"/> <peak value="458"/>
     </performance>
  </sdl:machine>
</wsdl:port>
```

When enhanced by this kind of meta-data, it is possible to use the WSDL specification to answer queries such as "is the service `SymbolicIntegrator` running on a machine with performance SPEC CINT2000 at least equal to or better than peak value 450?". To be able to answer queries such as "is there an implementation of the service `SymbolicIntegrator` using the package Mathematica?" we need to add an implementation description layer on top of the service layer.

---

[a] See `http://perseus.risc.uni-linz.ac.at:8080/openmath/index.html`
[b] `http://www.specbench.org/`

4 Implementation Layer

The next layer of description of a mathematical service collects the meta-data specific to the implementation. In a way similar to the service description, which has an abstract part and a concrete part, an implementation can be viewed as a collection of concrete realizations of an algorithm (with a problem specification) by a specific service. Each realization binds an algorithm or a problem to a service description. At the time of writing, we are drafting schemas for a mathematical implementation and problem definition language that will allow to specify implementation and problem descriptions such as the one here below.

```
<midl:definitions  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                   xmlns:midl="http://perseus.risc.uni-linz.ac.at:8080/msdl/midl">
                   xmlns:mpdl="http://perseus.risc.uni-linz.ac.at:8080/msdl/mpdl" ...>

  <import namespace="some-URI" location="some-URI"/>*
  <wsdl:documentation .... /> ?

  <midl:algorithm name="BubbleSort" >
      <mpdl:problem>
         <input name="nmtoken" .../>*
         <inputcondition .../>*
         <output name="nmtoken" .../>*
         <outputcondition .../>*
      </mpdl:problem>
  </midl:algorithm>

  <midl:servicebinding  name="BSortRiscBinding" type="BubbleSort">
    <wsdl:documentation .... />?  <-- extensibility element -->*

    <wsdlbinding:service portType="sort:BubbleSortPortType"
                         xmlns:sort=".../services/SortingService">
      <!-- address the exact operation for BubbleSort -->
      <invocation name="BubbleSortExec">*
          <wsdlbinding:operation name="sort:BSortOperation" .../>
      </invocation>
      <!-- concrete messages are related to inputs and outputs  -->
      <parameter name="nmtoken">*
          <wsdlbinding:message name="sort:some_messageName" .../>
      </parameter>
    </wsdlbinding:service>
  </midl:service>

  <midl:implementation name="BSort">
    <midl:realization name="riscBSort" servicebinding="BSortRiscBinding">*
      <midl:serviceaddress name=".../services/SortingService?wsdl" />
      <midl:bibliography xmlns:dc="dublin_core" .../>
      <midl:efficiencyfactor  .../>
      <midl:software .../>
    </midl:realization>
   </midl:implementation>

</midl:definitions>
```

As WSDL teaches, a specific XML language defined by a collection of XML schemata [12] is a suitable language for describing the functionality of mathematical services. Each problem class can be associated with a specific namespace defined by an XML schema, so for instance there is a schema for a computing problem. This schema acts as a specification of the basic skeleton structure of the problem instance. Problem descriptions will also be defined by XML schemas, as to associate a namespace to each specific problem, for instance a real root isolation problem could be described using the schema identified by the URI `http://www.mathbroker.net/computing/algebra/realrootisolation`. Schemas for problem instances include the schema of the problem class they belong to so that the class requirements on the descriptions are satisfied.

5 Conclusion and Work Plan

We have described the design of a language for the description of mathematical services in a distributed networked environment. The language is structured in layers, each one capturing a particular aspect of a mathematical problem solution, implementation and service details. This structured approach has the advantage of allowing a modular development of the language by starting from the bottom layer, which is in part covered by existing standard languages, and then adding onto it.

The final goal of the description language is to support the electronic processing of mathematical services in the scenario where the Net is a global problem solving framework where each computation can be accessed/found by a multitude of entry points.

Our current work is the exact definition of the XML schemas that will correspond to the layers, the identification of some case studies that should serve as a guideline, and the study of available on-line knowledge bases of mathematics to be used as reference. We have begun with analyzing available web services standards such as WSDL and related ones. Our experiments have convinced us that WSDL alone does not suffice if we want to eventually be able to reason about mathematical services, to streamline them and to do mathematics on-line.

Acknowledgement

This work has been supported by the Austrian Science Foundation (Project P15183).

References

1. Alessandro Armando and Tudor Jebelean, editors. *Special Issue on Calculemus-99*, volume 32 of *Journal of Symbolic Computation*. Academic Press, September 2001.
2. Paul Wang. Design and Protocol for Internet Accessible Mathematical Computation. In Sam Dooley, editor, *ISSAC'99 International Symposium on Symbolic and Algebraic Computation*, pages 291–298, Vancouver, Canada, July 28–31, 1999. ACM, New York.
3. JavaMath — Internet Accessible Mathematical Services, February 2001. `http://javamath.sourceforge.net`.
4. O. Caprotti, A. M. Cohen, H. Cuypers, M. Riem, and H. Sterk. Using OpenMath Servers for Distributing Mathematical Computations. In Wei Chi Yang, Sung-Chi Chu, and Jen-Chung Chuan, editors, *ATCM 2000: Proceedings of the Fifth Asian Technology Conference in Mathematics*, pages 325–336, Chiang-Mai, Thailand, December 17–21 2000. ATCM Inc., USA.
5. Stephane Dalmas, Marc Gaetano, and Stephen Watt. An OpenMath 1.0 Implementation. In Wolfgang Küchlin, editor, *ISSAC'97 International Symposium on Symbolic and Algebraic Computation*, pages 241–248, Maui, Hawaii, July 21–23, 1997. ACM Press, New York.
6. Patrick Ion and Robert Miner. Mathematical Markup Language (MathML) Version 2.0. W3C proposed recommendation, World Wide Web Consortium, January 2001. `http://www.w3.org/TR/2001/PR-MathML2-20010108/`.
7. Richard Zippel. The MathBus. In *Workshop on Internet Accessible Mathematical Computation at ISSAC'99*, Vancouver, Canada, July 28–31, 1999. `http://icm.mcs.kent.edu/research/iamc99proceedings.html`.
8. Andreas Franke and Michael Kohlhase. System Description: MathWeb, an Agent-Based Communication Layer for Distributed Automated Theorem Proving. In Harald Ganzinger, editor, *16th Conference on Automated Deduction*, number 1632 in Lecture Notes in Artificial Intelligence, pages 217–221. Springer, 1999.
9. A. Armando and D. Zini. Interfacing Computer Algebra and Deduction Systems via the Logic Broker Architecture. In *Proceedings of the XXth Calculemus symphosium*, St. Andrews, Scotland, 6-7 August 2000.
10. Arnon Avron. Simple consequence relations. *Information and Computation*, 92(1):105–140, 1991.
11. E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web

Services Description Language (WSDL) 1.1. W3C Note, 15 March 2001.
12. Paul V. Biron and Ashok Malhotra. XML Schema Part 2: Datatypes. W3C candidate recommendation, World Wide Web Consortium, October 2000. `http://www.w3.org/TR/2000/CR-XMLschema-2-20001024`.