From Untyped to Polymorphically Typed Objects in Mathematical Web Services

William Naylor, Julian Padget

Department of Computer Science, University of Bath, UK
{wn,jap}@cs.bath.ac.uk

Abstract. OpenMath is a widely recognised approach to the semantic markup of mathematics that is often used for communication between OpenMath compliant systems. The Aldor language has a sophisticated category-based type system that was specifically developed for the purpose of modelling mathematical structures, while the system itself supports the creation of small-footprint applications suitable for deployment as web services. In this paper we present our first results of how one may perform translations from generic OpenMath objects into values in specific Aldor domains, describing how the Aldor *interface* domain `ExpressionTree` is used to achieve this. We outline our Aldor implementation of an OpenMath translator, and describe an efficient extension of this to the Parser category. In addition, the Aldor service creation and invocation mechanism are explained. Thus we are in a position to develop and deploy mathematical web services whose descriptions may be directly derived from Aldor's rich type language.

1 Introduction

Mathematical web services are becoming an important feature in the web of today and it will likely be more so in the future. Computer algebra systems are one of the major technologies that can be used to support a certain class of mathematical web services. Here we show how the Aldor computer algebra system with its sophisticated type system, can be used both as a mathematical web service constructor and as a back end for mathematical web services. Several requirements fed the design and development of Aldor, amongst them being:

- **Interoperability**, so that integrating programs written in different languages is more straightforward and in particular, this makes it more amenable than most for writing mathematical web services due to the relative ease of integration with the inevitable Java
- **Strong typing** Aldor has a sophisticated two-level type structure based on *domains of computation*, that we will refer to as *domains* and *categories*. The type system is a direct descendant of that developed over many years in the line from Scratchpad through to Axiom, where the objective was to have a type language that was sufficiently tractable for checking—but not inference—yet rich enough to be able to capture the structure of mathematics. In Aldor, a domain is an environment providing a collection of exported constants including functions—analogous to a *class* in Java—while a category is used to specify information about a domain, in terms of a

collection of exports that the domain in question is required to provide—analogous to an *interface* in Java. Domains and categories in Aldor may be dependant on other Aldor objects that may be members of domains, domains themselves or categories; that is both domains and categories may be parameterized by other Aldor objects.

– **Efficiency**—both in speed and space—which is why Aldor provides a good basis for mathematical services, because Aldor programs may be compiled to provide executables with execution speeds comparable to that of C++. This allows services to be compiled prior to deployment and invoked with little overhead. Since the inception of Aldor—it was originally developed as a compiler language for the Axiom [2] computer algebra system in the early 1990s, however it has developed separately since—a number of stand-alone libraries have been developed, and specifically the *algebra* library [4] which provides a number of the domains and categories utilised by the work detailed in this paper.

A web service is not unlike a (remote) procedure in that the user must supply some inputs (arguments) and in return should receive some outputs (results). These input and output values will be represented in some communication language that it is desirable should not be system specific, because web services, so it is implied, should make no assumption about the context of the clients of the service. OpenMath[1] is our chosen language for the representation of mathematical objects for input to and output from the mathematical web services. OpenMath adopts a novel, but also historically enforced, solution to the unambiguous identification of objects, in that rather than using namespaces, which did not exist when OpenMath was first conceived, attributes indicate the referenced content dictionary and element. Thus `<OMS cd = "linalg2" name = "matrix"/>` in the example below identifies the `matrix` object in the `linalg2` content dictionary. The definitions are organised using Content Dictionaries (CDs) which may be stored in standard libraries, for example those maintained by the OpenMath society [12], or shared between applications. OpenMath may be represented in a number of ways but the accepted representation, especially as far as communication over the Internet is concerned, is in an XML (eXtensible Markup Language) [16] format.

Example 1. The matrix $\begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix}$ *may be represented in OpenMath markup as:*

```
<OMA>
  <OMS cd = "linalg2" name = "matrix"/>
  <OMA>
    <OMS cd = "linalg2" name = "matrixrow"/>
    <OMI>1</OMI> <OMI>3</OMI>
  </OMA>
  <OMA>
    <OMS cd = "linalg2" name = "matrixrow"/>
    <OMI>2</OMI> <OMI>4</OMI>
  </OMA>
</OMA>
```

where `linalg2` *is the second linear algebra content dictionary, defining the* `matrix` *and* `matrixrow` *concepts,* `OMA` *identifies an application (of a constructor in each of*

[1] We refer to OpenMath 1.0 in this paper as it is forward compatible and we do not require any of the improved features of the newer versions

the three cases here), `OMS` *identifies a symbol with attribute* `name` *in the given CD and* `OMI` *identifies an integer.*

The purpose of this paper is to demonstrate how the categories and domains of Aldor's type system can be used to advantage in the creation of mathematical web services and specifically, how it may be used to capture accurately the semantics of OpenMath input and subsequently how it assists in turning outputs back into OpenMath for communication to other OpenMath-aware applications.

We now outline the structure of the rest of the paper. In section 2 we describe the architecture of our service manager, discussing details of the problems that arose from the philosophical mismatch of Aldor's strong-typing and OpenMath's type agnosticism, and how we have overcome this by means of the `ExpressionTree` domain. In section 3 we discuss both the theoretical—from a type system point of view—and practical limitations of the approach we describe. In section 4 we give specific details of the service manager, how the services must be wrapped in order that they can communicate over the Internet and demonstrate how a service is invoked *via* Axis. Finally in section 5 we give an example of our service manager, demonstrating deployment of a service right through to service invocation and receiving OpenMath results over axis, which may then be converted *via* style-sheets to presentation MathML, which may subsequently be displayed by the browser.

2 Design of the Service Constructor Service

If a service is to be usefully deployed on the Internet, it must receive parameters, or input from the external world. It must also return the results of the computation to the external world. We utilise the conventional solution of a *wrapper* around the code that implements a service to provide the function of processing OpenMath received over an *Input Stream*. Consequently the OpenMath is converted into the internal Aldor representation that may then be processed by the service. On successful completion the service will produce results which are then converted to OpenMath and fed to an *Output Stream*. The input stream and output stream referred to in this description originate from a java class listening to and writing to an axis-created SOAP connection.

The first step in making Aldor OpenMath aware was the definition of an OpenMath domain. Fortunately, since there are already a set of domains implementing XML-DOM [17, 10] it was relatively straightforward to adapt these for the purpose. These domains take a domain-valued parameter that must have a *category* of character, such as `UTF8Char`. This expresses the constraint that there are a number of functions specified in the category that the domains must export. `UTF8Char` is the character domain which we use in this presentation, it being a full implementation of the Unicode UTF-8 encoding [15]. It is written to supply the full functionality of the character encoding and utilises the lowest level operations available in Aldor to ensure high efficiency.

2.1 Interface Typing Problems and our Solution

One of the major problems encountered when performing translations between OpenMath and Aldor is that to a large extent OpenMath is type agnostic. That is to say that

the objects appearing in OpenMath markup have no type information attributed to them; it is possible to give type information by means of `OMATTR` elements, however their use is not mandated by the standard and therefore we can not assume that received OpenMath markup will contain such attributes. Aldor values however are strongly typed. This means that all values, including those supplied as parameters to a function, must belong to a specific domain (or type), additionally the return value from a function must be of a specific type.

The service code supplied by its author specifies that its parameters and its return value are of specific types, and the parameters received over the input stream and the return value to be sent to the output stream have to be expressed in OpenMath. This requires that there is some translation mechanism between the heterogeneous type structure of Aldor and the single OpenMath domain. Furthermore, in order that the wrapper generator is well-structured and extensible, it is necessary to present the functionality via a carefully-defined API. We propose a solution to this problem in section 2.4 and have built a service generator based on it, which is detailed in section 4. There are a number of categories and domains which are central to our solution which we shall detail below.

2.2 ExpressionTree

The purpose of this section is to explain how to translate between Aldor internal objects and other (external) representations. The key to the translation process is the `ExpressionTree` domain, which acts as a gateway to a number of external representations, *e.g.* TeX, C, Fortran, lisp or maple, and specific Aldor domains. Most of the domains in Aldor satisfy the `ExpressionType` category[2], that is they export a function with the following signature:

```
extree : % -> ExpressionTree;
```

This should be read as meaning that the domain exports a function with name `extree`, which takes a parameter of type `%` (the domain in question) and returns a value of type `ExpressionTree`.

Example 2. The Aldor domain `DenseMatrix(R)` *represents matrices of elements of type* `R` *in a dense format, the parameter* `R` *must be a type which has both the categories* `ArithmeticType` *(meaning one can perform arithmetic operations on its elements), and* `ExpressionType` *(one can make* `ExpressionTree` *objects from its elements). The domain* `DenseMatrix` *is of the* `ExpressionType` *category, this means that it has an* `extree` *function which can construct* `ExpressionTree` *objects from its objects, as seen in the following Aldor interpreter snippet:*

```
%1 >> mat:DenseMatrix(Integer) := [[1,2],[3,4]]
matrix [[1,3],[2,4]] @ DenseMatrix(AldorInteger)

%2 >> exmat := extree(mat)
(matrix 2 2 1 3 2 4) @ ExpressionTree
```

[2] The categories `ExpressionType`, `Parser` and `Parsable` actually require exportation of additional functions, however we are only concerned with the ones mentioned.

the presentation given after the second interpreter statement should be read as: (i) The first value indicates the type of object represented, (ii) The second two values are the dimensions of the matrix and (iii) the rest are the values in the matrix. One also notes that the type `AldorInteger` *is also of category* `ExpressionType`. *Now that we have an* `ExpressionTree` *version of our* `DenseMatrix` *objects, we are in a position where we can transform them into a number of external formats, as follows:*

```
%3 >> axiom(stdout,exmat)   -- Axiom format
matrix [[1,3],[2,4]]  () @ TextWriter

%4 >> maple(stdout,exmat)   -- maple format
linalg[matrix](2,2,[1,3,2,4])  () @ TextWriter

%5 >> tex(stdout,exmat)     -- tex format
\pmatrix{
1 & 3 \cr
2 & 4\cr }   () @ TextWriter
```

The above deals with converting values from a specific Aldor type into a generic Aldor type, which may then be converted into a number of different external types. For our needs we must convert to the `OpenMath` type. We may achieve this by extending the `ExpressionTree` domain with a function which has signature:

```
openmath: (TextWriter, %) -> TextWriter
```

Which performs the task of converting `ExpressionTree` objects into OpenMath. This enables Aldor to communicate Aldor values to the external world in an unambiguous machine processable manner.

To communicate in the opposite direction, it is necessary to: (i) convert from the external markup to `ExpressionTree`, in our case from OpenMath to `ExpressionTree`, then (ii) from `ExpressionTree` to the specific Aldor domain. Implementation of step i) involves extending the OpenMath domain with the `Parser` category that is described in more detail in section 2.3. Step ii) requires that the target domain must be of category `Parsable`, which means that the domain must export the function with signature:

```
eval: ExpressionTree -> Partial(%)
```

This should be read as meaning that the domain exports a function `eval` which takes an `ExpressionTree` argument and returns a value of type `Partial(%)`, which is Aldor's name for lifted domains, that is domains extended by $\perp$. Aldor's name for $\perp$ is `failed`, indicating that the expression tree does not represent an object from the domain referred to by `%` (*i.e.* the domain in question), or it may be a value from the domain referred to by `%`. In the later case a value of type `%` may be obtained using the function `retract` exported by the `Partial(%)` domain.

Example 3. We shall continue example 2, and show how we can reconstruct the matrix from the `ExpressionTree` *representation. (N.B. during execution of a service the* `ExpressionTree` *objects to be evaluated will originate from* `OpenMath` *objects.)*
First we translate from `ExpressionTree` *to* `Partial(DenseMatrix(Integer))` *using the* `eval` *function, available from the* `DenseMatrix` *domain since this domain is of category* `Parsable`.

```
%6 >> pmat:Partial(DenseMatrix(AldorInteger)) := eval(exmat)
[F matrix [[1,3],[2,4]]] @ Partial(DenseMatrix(AldorInteger))
```

Finally we translate from the `Partial(···)` *domain to the specific domain:*

```
%7 > retract(pmat)
matrix [[1,3],[2,4]] @ DenseMatrix(AldorInteger)
```

2.3 Parsing OpenMath in Aldor

In this section we describe the approach we have taken in extending the OpenMath domain with the `Parser` category. In order that the OpenMath domain satisfies the `Parser` category, it is necessary that the OpenMath domain exports a function with signature:

```
parse! : % -> ExpressionTree
```

that is, export the function `parse!` which takes an `OpenMath` value (the specialisation of `%` in this case) and returns the `ExpressionTree` equivalent. The naïve approach might simply traverse a table of OpenMath classes, associating one class of OpenMath objects with its `ExpressionTree` equivalent. Although, this approach would certainly work, the complexity would depend on the size of the table which would be large. The complexity would become worse as the number of OpenMath objects handled became large (*i.e.* it would not scale well). The algorithmic complexity of this process would be $O(nm)$ where n is the number of elements in the document and m is the number of different classes of OpenMath objects handled.

The approach that we have taken is to build up a hash table associating strings characterising OpenMath objects with functions taking an OpenMath object as parameter and returning `Partial(ExpressionTree)` objects. The functions may be extracted from the hash table dynamically and applied during a recursive descent of the OpenMath XML tree. The algorithmic complexity of this operation will be $O(n)$ where n is the number of child elements in the document.

Example 4. If we are translating the OpenMath element: `<OMI>10</OMI>` *into Aldor, the tag-name "OMI", is used as the key to the hash table; this is associated with a function which takes an OpenMath object as parameter and returns an* `ExpressionTree`*. This particular function takes the content of the* `OMI` *element (*10 *in this case), and returns its* `ExpressionTree` *representation.*

Example 5. If we are translating the OpenMath application element:

```
<OMA>
  <OMS cd="set" name="set1"/>
  ...
</OMA>
```

elements are members of the set being constructed. To characterise this element we concatenate the values of the `cd` *and* `name` *attributes with an "@" separator, to obtain "set@set1"*[3] *in this case. The value obtained from the hash table will be a function which we apply to the OpenMath object. The body of this function recursively performs a* `parse!` *application on all the children, bar the first, of its argument to obtain their values as* `ExpressionTree` *objects. Consequently, we are in a position to build an* `ExpressionTree` *set and return this as the return value of the function.*

[3] The characterisation used is implementation specific, we report the one that we have used.

2.4 Solving the Type Translation Problem

Summarising our solution to the type translation problem, our method performs the following steps:

1. Read characters from the input stream (this is assumed to be OpenMath XML, if not an error will be returned to the client),
2. Convert the input stream to internal Aldor `OpenMath` objects,
3. Convert the `OpenMath` objects to `ExpressionTree` objects *via* the technique detailed in section 2.3,
4. Parse the `ExpressionTree` objects to be of type `Partial(S)` where `S` is the specific type of the particular parameter, for this step to be possible the parameter type must be of the category `Parsable`,
5. The `Partial(S)` object so obtained must now be retracted to the type `S`, in which format it may be processed by the functions provided by the service author,

After the service code has been executed, a return value will be generated with a single return type, this must be converted to OpenMath, in order that it may be sent back to the client over axis.

6. The return type must be of category `ExpressionType`; this implies that the relevant domain exports the `extree` function which implements the transformation to `ExpressionTree`,
7. Call the `openmath` function with which the `ExpressionTree` domain has been extended, in order to convert the `ExpressionTree` to OpenMath format and write it to the standard output.

3 Theoretical and Practical Limitations

Both the Aldor system and the OpenMath markup language are extensible. This means that one may define new domains in the former allowing one to construct novel objects to interact with the rest of the system. We may then write new OpenMath Content Dictionaries, which define new symbols allowing one to represent these objects. This extensibility has its advantages and drawbacks. On the one hand this means that if a service is utilising objects that are not handled by the system, there is a possibility that the system may be extended to deal with them, however fundamental limitations surely exist and we must determine the limitations on the scope of the system. In this section we consider the limitations on the system.

3.1 Theoretical Limitations

The basic limitations on the types of objects which may be accepted by the services are that it must be possible to perform the required translations. These limitations naturally fall into the following partitions:

Category considerations: The type of the parameters must have the category `Parsable`, in order that the parameter objects may be obtained from their `ExpressionTree` formats. This is in order that step 4) in section 2.4 may be performed. The return type must be of category `ExpressionType` in order that an `ExpressionTree` object may be obtained from the return value (step 6), section 2.4). If any of these domains do not have the required category this may be rectified using Aldor's `extend` facility, which allows extra functions to be exported by a domain.

Translation from `OpenMath` to `ExpressionTree`: To effect the translation from `OpenMath` objects to `ExpressionTree` objects, we use the technique detailed in section 2.3. The `HashTable` which it utilises must be loaded with the correct functions, otherwise there is no information to specify how the transformation is to take place. In our current prototype, the set of translation methods is fixed. Clearly this is not practical for the longer term, due to the extensible nature of OpenMath which means that new CDs will be written and new translation methods will be required. We are currently considering how this may be made more flexible within the constraints and capabilities of Aldor.

Function objects: Greater problems arise with translation to and from function objects and the remainder of this section is on-going work. In Aldor function objects are treated as first class objects which may be assigned to variables, passed as parameters and returned from functions. In OpenMath abstract functions may be represented using a λ-binding notation:

Example 6. The function which a mathematician might write as: $\lambda\, x \cdot x^2$ could be represented in OpenMath as

```
<OMBIND>
  <OMS cd="fns1" name="lambda"/>
  <OMBVAR><OMV name="x"/></OMBVAR>
  <OMA>
    <OMS cd="arith1" name="power"/>
    <OMV name="x"/>
    <OMI>2</OMI>
  </OMA>
</OMBIND>
```

Here are some of the issues this matter raises:

- The first problem is that function objects are not members of domains, and so the extend mechanism referred to earlier can not be used. It is envisaged that a special purpose *package* would be written to translate from objects of the `OpenMath` domain to objects of `ExpressionTree`, and thence to the primitive function objects.
- A second problem becomes apparent with this approach: it appears that the `ExpressionTree` framework does not supply a rich enough descriptive mechanism to describe function objects. It is hoped that this framework could be extended in some way. A different approach might be to translate directly from `OpenMath` to the primitive function objects.
- A third and apparently intractable problem arises from the translation from Aldor to OpenMath, since this would imply decomposing the function objects into their constituent parts. Currently no tools exist in Aldor to do this and we must leave this as future work.
- However, there is some hope: if the OpenMath input were to contain fully annotated types—this is not the normal practice—this information could be propagated through the process and potentially re-exported when needed. Indeed, annotation is probably the only way forward, given the computational intractability of type inference for type systems such as that in Aldor.

A number of the objects which exist in the algebra library have function objects as part of their representation, *e.g.* `DenseUnivariateTaylorSeries`, it is not possible to represent these objects without using functions. For the above reasons we are not currently able to translate between these objects and OpenMath.

Algorithm 1 Service Wrapper

serviceWrap():() == {
  $\{x_1, \cdots, x_n\}$ $\leftarrow$ read OpenMath arguments from the default Input Stream
  $\{E_1, \cdots, E_n\}$ $\leftarrow$ convert $\{x_1, \cdots, x_n\}$ to `ExpressionTree`
  ret:R $\leftarrow$ service_code$(E_1, \cdots, E_n)$ — R is the return type of the service code
  return openmath(stdout,extree(ret))
}

service_code$(E_1$: `ExpressionTree`$, \cdots, E_n$: `ExpressionTree`$)$:R == {
  import from `Partial(D`$_1$`)` $\cdots$ `Partial(D`$_n$`)` and `D`$_1$ $\cdots$ `D`$_n$, where `D`$_1$ $\cdots$ `D`$_n$ are the arguments to the service code.
  $\{e_1, \cdots, e_n\}$ $\leftarrow$ convert the `ExpressionTree` objects into objects of the specific types.
  the rest of the service code }

4 Building and Using Aldor Web Services

We have built a web service manager which supports the construction and deployment of Aldor-based web services and is achieved via a set of JSP pages. One of the functionalities of the manager is to assist users in the deployment of services by writing the generic web services code automatically. The user simply supplies the code that implements the service and the web service manager sends this code as a string through Axis to a wrapper service. We built a similar system earlier as part of the MONET project [9, 6] for the deployment of Maple functions as web services.

4.1 The Wrapper Service

The purpose of this wrapper service is to convert the function or functions which implement a service into a set of functions which take their parameters as OpenMath objects from an input stream and convert them into the required type for the function, similarly it will translate the return value of the function into OpenMath and write that to the output stream. To do this we must translate the code submitted by the service implementer into code to perform the actions outlined in Algorithm 1.

Construction of wrapped services is performed dynamically as the service code is received, since the specific details of the wrapped code will depend on the type of the arguments and return values of the code. We perform this service wrapper creation using a java program which implements the actions detailed in Algorithm 2.

4.2 Service Invocation

The service manager also allows invocation of the wrapped web services as follows:

1. The client selects which service they wish to invoke,
2. The client is prompted for the requisite number of parameters of the required types,
3. The client supplies these parameters in OpenMath format. It would be possible to perform type checking at this stage,
4. The request is made *via* a SOAP [14] engine, for example Axis,

Algorithm 2 Wrapper Creation

input: `prog` – The service code
Extract the arguments and their types from the *interface function* of `prog`.
if The argument types are not of category `Parsable` **then**
throw a TypeNonParsable exception, whose detail records the types which are not of category `Parsable`(see note below).
end if
Extract the return type of the *interface function* of `prog`.
if The return type is not of category `ExpressionType` **then**
throw a TypeNonParsable exception
end if
Build the program detailed in Algorithm 1 where the arguments are those given as the parameters of `prog`
Compile the constructed program, and store the executable in the service database.

Note: The type information recorded by the exception may be useful to the implementers of the service manager, as they will then be given information about extensions required by service implementer clients.

5. The wrapped service, as described in section 4, is invoked and supplied with the OpenMath parameters,
6. The service does the required processing and transmits OpenMath results which are returned (over Axis),
7. The client receives the OpenMath results, it may do further processing, *e.g.*, translation to Presentation MathML using XSLT style-sheets.

4.3 Automatic MSDL Generation

After a service has been created, it is necessary to construct its advertisement as a web service that may be stored in a UDDI-like repository and subsequently found by service brokers. A straight Web Service Description Language description is relatively unhelpful since it only contains the information necessary to invoke the service. Extended UDDI registries contain textual descriptions, but these too are of little use for software clients. These deficiencies were essentially the motivations behind the MONET project's development of Mathematical Service Description Language (MSDL) [5] that takes some inspiration from its contemporary DAML and DAML-S by describing a service in terms of pre-conditions and post-conditions. In the XML markup used to represent the MSDL document these are represented by the following elements:

- `input` elements, the signatures of the input parameters,
- `output` elements, the signatures of the return values,
- `pre-condition` elements, conditions which must hold prior to service execution and
- `post-condition` elements, conditions which must hold after the service has executed

In both MONET and GENSS[4], it has been necessary to construct the MSDL descriptions mostly by hand, which is an arduous and error-prone task. A particular benefit of the capacity to translate between OpenMath and Aldor's type system is that it is possible to generate, as a side-effect of Algorithm 2, the types of the arguments to the service and the type of the return value. These values may be used to create automatically the `input` and `output` elements respectively and also some of the basic constraints for the pre- and post-conditions. The association

[4] GENSS is a follow-on project to MONET; see `http://genss.cs.bath.ac.uk`

```
<monet:definitions
  targetNamespace= "http://monet.nag.co.uk/problems/">
<monet:problem name ="AntiTranspose">
  <monet:header>
    <monet:taxonomy taxonomy= "http://gams.nist.gov" code="GamsD1b"/>
  </monet:header>
  <monet:body>
    <monet:input name ="M">
      <monet:signature>
        <om:OMOBJ>
          <om:OMA>
            <om:OMS cd ="sts2" name ="matrix"/>
            <om:OMS cd="setname1" name="Z"/>
          </om:OMA>
        </om:OMOBJ>
      </monet:signature>
    </monet:input>
    <monet:output name ="A">
      <monet:signature>
        <om:OMOBJ>
          <om:OMA>
            <om:OMS cd ="sts2" name ="matrix"/>
            <om:OMS cd="setname1" name="Z"/>
          </om:OMA>
        </om:OMOBJ>
      </monet:signature>
    </monet:output>
    <monet:pre-condition>
      <om:OMOBJ>
        OpenMath for the number of columns in A = the number of rows in A
      </om:OMOBJ>
    </monet:pre-condition>
    <monet:post-condition>
      <om:OMOBJ>
        OpenMath for A_rc = M_{len-c+1,len-r+1} where len is the size of the matrix
      </om:OMOBJ>
    </monet:post-condition>
  </monet:body>
</monet:problem>
</monet:definitions>
```

Fig. 1. MSDL generated from the antiTranspose example (see Section 5)

of this information with a service is important not only for service advertisement [11], but also in checking the correctness of parameters during a service invocation.

5 Example

We assume that an Aldor service has been deployed through Axis. The service we demonstrate is one that calculates the *Anti-diagonal* of a matrix *viz.* the matrix obtained by reflection about the anti-diagonal of the matrix. The code to implement calculation of the anti-diagonal:

```
antitran(m:DenseMatrix(Integer)):DenseMatrix(Integer) == {
  import from MachineInteger,Integer;
  ret := copy m;
  len:MachineInteger := numberOfColumns m;
  for c in 1..numberOfColumns m repeat {
     for r in 1..numberOfRows m repeat {
        ret(r,c) := m(len-c+1,len-r+1);
     }
  }
  ret
```

may be submitted to the service manager *via* the JSP page shown in figure 2. Additional information, apart from the code, which must be associated with the service are its description for the purpose of service advertisement, discovery *etc.*. The details of the interaction is shown in Fig 3.

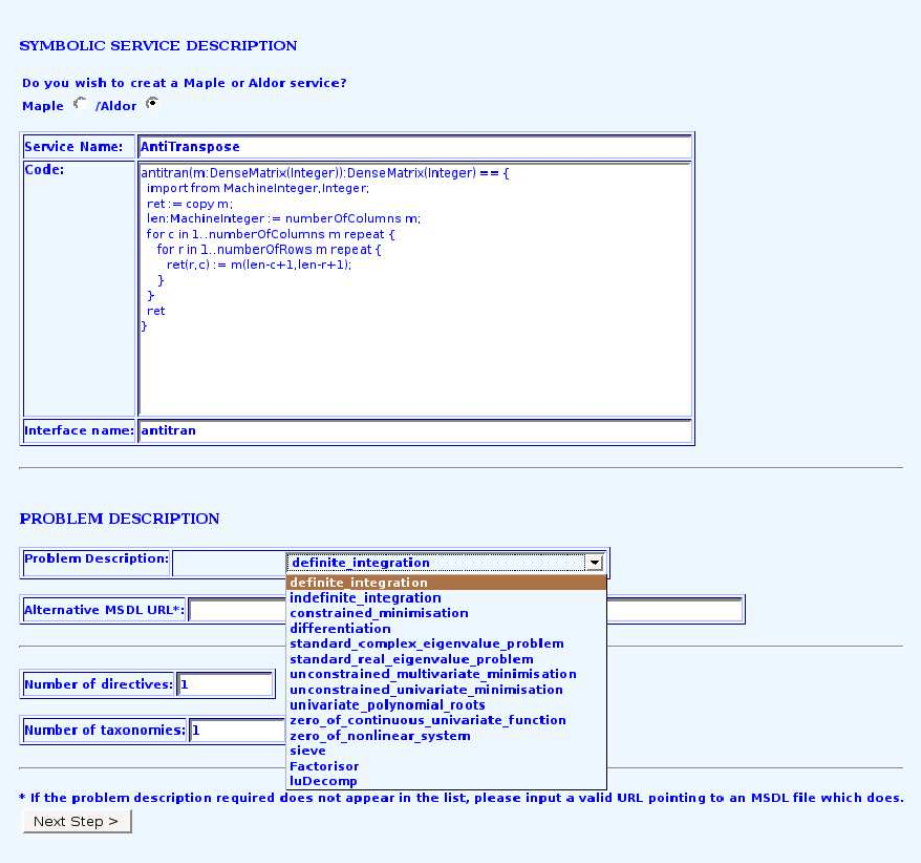


Fig. 2. Submission of Code to the service manager

6 Related Work

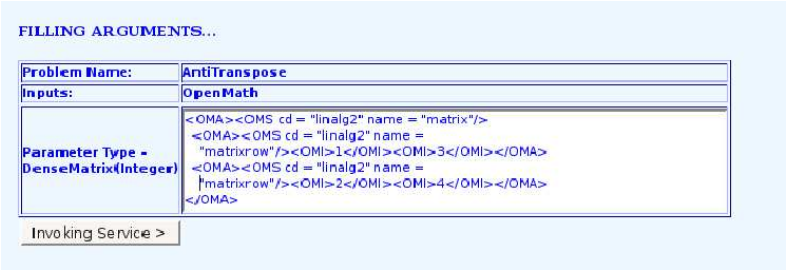
The work detailed in this paper utilises a similar architecture as that used by the service manager used in the GENSS project [8]. The services created as part of GENSS were based on the Maple computer algebra system. Conversion from OpenMath to Maple is less problematic then conversion between OpenMath and Aldor, due to the fact that Maple, being a latent-typed language, can better accommodate OpenMath's type agnosticism. This conversion however is performed by procedures written in the Maple interpreted language [13] with the relatively high overhead involved in constructing a new Maple instance every time a service is invoked. Indeed, it was the time taken in launching Maple and loading all the necessary library code that drove us to seek an alternative solution, resulting in the application of Aldor reported here. The GENSS project was a follow-on project from the EU funded MONET project under which the MSDL [5] and OpenMath based ontologies which are the underlying communication languages for these projects where developed. The Maple based service manager created in the GENSS project followed a similar method to that described by Dewar et al. [7] where the service manager developed at the University of Western Ontario is described, along with associated technologies. The MathBro-

INTERACTING WITH AN ALDOR WEB SERVICE

When invoking a service, the interface first invites the user to select a service from the list of services which the manager has in its database. In order to deal with the case where different instances of a service have been deployed, with or without the same implementation, the service manager associates a unique identification number with each service.

For this naïve implementation of the manager, we use a string representation of the OpenMath parameters. A more user friendly approach would be to offer several alternative representations, *e.g.* Aldor, Mathematica, Maple format then issue a call to a translation service which would translate the parameters to OpenMath, This is a trivial extension, but is outside the scope of the current work. Returning to the example, the parameter supplied is the matrix

$$\begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix}$$



The service is then invoked over Axis. This in turn invokes the Aldor executable created as outlined in section 4.1. The OpenMath parameters are sent to the service by a Java method over and Output Stream. This same method receives the input from the executable on an Input Streams as OpenMath, which is transmitted by axis back to the client, where the JSP page converts the OpenMath to Presentation MathML, and the following page results:

$$\begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix}$$

$$\begin{pmatrix} 4 & 3 \\ 2 & 1 \end{pmatrix}$$

Fig. 3. Phases of interaction with the Aldor-based web service

ker I and MathBroker II projects have focused on mathematical service brokerage [3], apparently using taxonomic information to perform service categorization and selection. It would be interesting to investigate how the MathBroker software might be used as advertisement agencies for Aldor-based services.

7 Conclusions

We have described how the Aldor language, and in particular its type system, can be used to advantage in the creation and publication of mathematical web services. By creating translators between OpenMath and Aldor objects for (a subset of) the Aldor type system, and *vice-versa*, we are able to deploy Aldor-based web services that accept OpenMath as input and generate OpenMath as output, through the use of wrapper service code. Both the wrapper code and the service code may be compiled and stored in a repository. A particular challenge of the work has been accommodation of the tension between OpenMath's absence of need for type information and Aldor's sophisticated strong typing scheme, but by using the `ExpressionTree` domain, and an efficient, linear time algorithm for the translation of OpenMath objects to Aldor objects, it is possible to handle the different Aldor objects that may have very different procedures for providing this translation. Furthermore the correspondence between Aldor types and OpenMath that has thus been established can now be applied to the signature of the service procedure and used to generate some of the necessary information to go in the MSDL description that might subsequently enable a broker to identify the service as useful.

Acknowledgements The work reported here was partially supported by the Engineering and Physical Sciences Research Council under the Semantic Grids call of the e-Science program (grant reference GR/S44723/01).

References

1. Web Services - Axis, Apache Project. Available *via* `http://ws.apache.org/axis/` (March 2006).
2. The Axiom Computer Algebra System. Available *via* Wiki at `http://wiki.axiom-developer.org/FrontPage` (March 2006).
3. Rebhi Baraka, Olga Caprotti, and Wolfgang Schreiner. A Web Registry for Publishing and Discovering Mathematical Services. In *EEE*, pages 190–193. IEEE Computer Society, 2005.
4. Manuel Bronstein and Marc Moreno Maza. The Standard Aldor Library, Version 1.0.2. Available *via* `http://www-sop.inria.fr/cafe/Manuel.Bronstein/libaldor/html`.
5. Stephen Buswell, Olga Caprotti, and Mike Dewar. Mathematical Service Description Language. Technical report, 2003. Available from the MONET website: `http://monet.nag.co.uk/cocoon/monet/publicdocs/monet-msdl-final.pdf`.
6. Olga Caprotti, Michael Dewar, James Davenport, and Julian Padget. Mathematics on the (Semantic) Net. In Christoph Bussler, John Davies, Dieter Fensel, and Rudi Studer, editors, *Proceedings of the European Symposium on the Semantic Web*, volume 3053 of *LNCS*, pages 213–224. Springer Verlag, 2004.
7. Mike Dewar, Elena Smirnova, and Stephen Watt. XML in Mathematical Web Services. In *XML Conference proceedings*, 2005.
8. GENSS Home Page. Available from `http://genss.cs.bath.ac.uk` (March 2006).
9. MONET Home Page. Available from `http://monet.nag.co.uk` (March 2006), 2002.

10. William Naylor. The XML-DOM domain for the Aldor computer algebra system. Available *via* `http://www.cs.bath.ac.uk/~wn/Papers/usersGuide.ps`, (March 2006).
11. William Naylor and Julian Padget. Semantic Matching for Mathematical Services. In *Proceedings of the Conference on Mathematical Knowledge Management 2005*, volume 3863 of *LNAI*. Springer Verlag, 2005. In press.
12. OpenMath website. `http://www.openmath.org`, February 2002.
13. Clare So, Zheng Wang, Sandy Huerter, and Stephen Watt. An Extensible OpenMath-Maple Translator. In *East Coast Computer Algebra Day (ECCAD) 2004*. Wilfred Laurier University, Waterloo, Ontario, 2004.
14. SOAP – Simple Object Access Protocol. Available *via* `http://www.w3.org/TR/soap/` (March 2006).
15. The Unicode Standard. Available *via* `http://www.unicode.org/standard/standard.html` (March 2006).
16. Extensible Markup Language (XML), W3C. Available *via* `http://www.w3.org/XML/` (March 2006).
17. Document Object Model (DOM) Level 2 Core Specification, W3C. Available *via* `http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113/` (March 2006).