Mathematical Software: The Next Generation?

Mike Dewar & David Carlisle
The Numerical Algorithms Group Ltd.
Wilkinson House, Jordan Hill Rd, Oxford, OX2 8DR
{miked,davidc}@nag.co.uk

1 Introduction

For more than ten years, there have been regular predictions of radical changes in the design of mathematical software. Conventional programming languages in general and Fortran in particular were dead, and interactive packages would only survive if they adopted open protocols and became more component-based. These predictions have so far failed to come true and the market for libraries such as NAG on the one hand, and packages such as Matlab, Maple and Mathematica on the other, continues to expand. At the same time mathematical and statistical technology has become more pervasive, often embedded inside application software packages which allow people who have no formal mathematical training to perform sophisticated computations which a few years ago would have required the services of a specialist.

One reason why existing software continues to be popular is that longevity and a good reputation amongst users are the most reliable indicators of correctness, a state of affairs which is liable to remain true for some time. Another important factor is economics: re-creating any major piece of mathematical software from scratch would almost certainly not be commercially viable. Finally, the learning curve in using any complex software can be quite great and users are reluctant to move away from a familiar combination of syntax and semantics. Thus the challenge for mathematical software developers is not so much how to write programs that make use of the latest techniques in software engineering, but how to make their existing software accessible to users in their preferred environments.

In the light of this experience it is tempting to be sceptical and predict that change will continue to be gradual. We will argue however that for both technological and social reasons the technology behind the semantic web has the potential to transform our industry.

2 Scientific Software

From NAG's perspective as both a producer and vendor of scientific software, we can categorise our user-base or market (depending on your point of view) very broadly as follows:

Research & Investigation This group of users typically employ a range of pieces of software on a daily basis. They might work in a university department doing pure research but they could equally be employed in a bank predicting future market trends. They need access to a range of tools, either acquired off-the-shelf or made-to-measure, and benefit from access to the state of the art.

Production This group of users employ specific software on a regular basis, e.g. as part of quality control, to direct a robot on a manufacturing line etc. The software is most likely created specially or at least tailored for their specific application.

Education & Training A significant group of users who traditionally use mathematical software packages as an adjunct to a conventional lecture-based course, but who in future might be offering training at a distance across the internet.

In what follows we focus on the requirements of the first two groups.

Different mathematical software systems often have a very steep learning curve and even systems designed for the same kinds of applications (e.g. Maple and Mathematica) can employ completely different programming styles not to mention syntax. A "holy grail" for scientific software since at least the 1980s (see for example some of the papers in [5]) has been to devise problem solving environments which enable a user access several packages via a common front-end, in the same way that a modern browser might launch different helper applications to process data of different kinds. There are at least three fundamental problems with this however:

1. the semantics of both the objects and the algorithms which one finds in a piece of scientific software are usually subtle and rarely explicit;
2. many mathematical operations involve exception conditions which need to be handled appropriately;
3. in practice users need to be able to trace or debug a computation to understand why they got a particular answer and this is difficult in a non-homogeneous software environment.

If these problems could be overcome then the advantages are clear. Users would no longer be "locked-in" to a particular software package and could access a wide range of functionality almost instantly via their preferred interface. The current situation where users have to wait months for new releases containing bug-fixes would potentially be eliminated with the latest versions of software being accessible either remotely via the web or downloaded and installed on demand. Users would find it easy to compare software and developers would be able to specialise.

Solving the above problems and devising an infrastructure which is usable in practice rather than merely of academic interest is non-trivial, but we believe that many of the ideas and technologies which underpin the semantic web can help us to achieve this. In particular the notion of so-called *intelligent agents* which can carry out tasks which might normally require the intervention of a human expert, and support for explaining decisions and the sequence of computations which lead to a particular result in a coherent and comprehensible way, are essential ingredients.

What we envisage is a flexible framework where various services would be accessible to a user and, indeed, to each other. A researcher would probably access them via a familiar user interface and would not normally care about whether e.g. an answer was determined by table look-up or by algorithmic computation. However, just as in some existing (homogeneous) environments, it would be possible to find out how an answer was determined and ask for a justification of the method used. In selecting a solution method the software might use the user's own preferences, or seek

"advice" as to which systems were best. Human users regularly post to Usenet or apply to software vendors for trial versions of software to road-test them in the context of their particular application area. Agents might amass data about the quality of results derived from different services or send the same problem to several services.

Somebody developing production software might start off this way but at some point would want to fix the particular software used and perhaps ensure that it was all available on a particular local machine. This would ensure reproducibility of results, and a reliable and predictable service. This flexibility and scalability is just as critical an aspect of such a framework as its convenience.

In the next few pages we will look at what technologies exist to help achieve this goal, and what still needs to be developed.

3 Representing Mathematical Objects

We said earlier that the precise semantics of a mathematical object in a given software system are often hard to tie down. Users may not even be aware of the different ways in which two systems might interpret the same input, nor will they care if the end result is correct or at least in line with their expectations. However if we are to exchange objects between different software packages then of course they need to agree the data formats to be used.

Protocols already exist which allow semantically-rich representations of mathematical objects to be exchanged between applications, in particular OpenMath [8] and Content MathML [6]. Introductions to both OpenMath and MathML can be found in [3]. Once a user has specified the semantics of an object, a package reading that object ought to preserve or at least respect those semantics.

As an example of different but equally valid interpretations of the same input in two similar systems, consider the string "2∗x+1". Typing this into Maple 6 yields an object equivalent to the OpenMath expression:

```
<OMOBJ>
  <OMA>
    <OMS name="plus" cd="arith1"/>
      <OMA>
        <OMS name="times" cd="arith1"/>
        <OMI>2</OMI>
        <OMV name="x"/>
      </OMA>
    <OMI>1</OMI>
  </OMA>
</OMOBJ>
```

where the semantics of the OpenMath Symbols (OMSs) `plus` and `times` are defined in the *content dictionary* called `arith1`, and their arguments are the variable `x` and the arbitrary precision integers 1 and 2. In other words this is an expression tree. Typing the same expression into AXIOM constructs an object equivalent to:

```
<OMOBJ>
  <OMA>
    <OMS name="polynomial_r" cd="polyr"/>
    <OMA>
      <OMS name="polynomial_ring_r" cd="polyr"/>
      <OMS name="Z" cd="setname1"/>
      <OMV name="x"/>
    </OMA>
    <OMA>
      <OMS name="poly_r_rep" cd="polyr"/>
      <OMV name="x"/>
      <OMA>
        <OMS name="term" cd="polyr"/>
        <OMI> 1 </OMI>
        <OMI> 2 </OMI>
      </OMA>
      <OMA>
        <OMS name="term" cd="polyr"/>
        <OMI> 0 </OMI>
        <OMI> 1 </OMI>
      </OMA>
    </OMA>
  </OMA>
</OMOBJ>
```

Without going into details of the representation, this is the recursive multivariate polynomial $2x+1$ whose coefficients are integers. Both expressions display the same in the different systems and some simple operations such as *what is the leading coefficient?* will give the same answer. However an operation such as adding the floating-point number 1.0 will give different results in the two systems. By making the semantics explicit (as in the above OpenMath expressions) we remove any ambiguity and so it would be wrong for AXIOM to represent the first object as a polynomial or for Maple to represent the second object as an expression tree.

A common interchange format which makes semantics explicit is an important ingredient for our mathematical software framework, but it only solves part of the problem. We need to be able to specify what we want to do with an object and, in all but the most trivial cases, map that abstract specification into a program in our mathematical server. AXIOM and Maple are designed to solve similar kinds of problems and in many cases implement the same algorithms, but the sequence of steps which a user has to take to solve a given problem will often be different. In OpenMath the piece of software that does this is called a *phrasebook*.

4 Describing Mathematical Interfaces

Many mathematical operations can be viewed naturally as simplifications or transformations, for example $\int_0^t cos(x)\,dx \rightarrow sin(t)$. However the algorithms which implement these operations may require extra information to be provided by the user: parameters which control the way the software itself operates, and resources to allow the software to do do its job (e.g. a workspace array for a Fortran routine, permission to write to a file or display on a particular machine in a distributed

environment ...). As an example, a routine in the NAG Fortran Library which is designed to solve integrals of the above type (D01ANF) requires the user to provide acceptable tolerances for both the absolute and relative accuracy of the result, an indication of the action to take if the algorithm fails to compute an acceptable solution, as well as two arrays to be used as temporary workspace.

Many component-based systems have a way of providing a *functional* description of the interface to a component, perhaps the best knowledge of which is the Object Management Group's *Interface Description Language* (IDL) which in particular is used in many implementations of CORBA and in Microsoft's COM-derived technologies (DCOM, ActiveX etc.). This provides a platform-independent description of an interface in terms of data types, intent (input/output etc.), and whether an object is private or shared, which can then be used to generate the extra software needed to connect to or embed the component.

From a purely mechanical point of view, an IDL description of a piece of algorithmic software does allow it to be used in a client-server setting. However experience in embedding NAG software in a variety of environments has shown that, in practice, this is not enough. In addition, there is more to providing a usable interface than just linking two bits of software together. NAG provides extensive documentation about each algorithm and parameter, advice to users about what to do if the algorithm fails, and examples of use. In some contexts there is also advice on which algorithm to use and on how to combine algorithms (e.g. scaling or pre-conditioning a problem). Some parts of this documentation are language specific (e.g. Fortran syntax is used, arrays are 1-based, ...) while other parts are generic. Finally there is also a test suite designed to verify the correctness of the software with different compilers on different platforms.

NAG is currently developing a framework which includes a detailed XML-based specification of the interface to an algorithm and of its documentation. Parameters are classified according to whether they carry data, control the way the algorithm works or are artifacts of the Fortran language. Constraints and default values are encoded in a structured, parseable form, and all language-specific parts of the documentation are marked-up to enable them to be transformed appropriately.

Early experiments suggest that this system is sufficient to allow us to generate documented interfaces to our existing Fortran code for a variety of environments. For example, NAG might offer a service to Mathematica users where they were provided with the necessary material to use certain NAG routines from within Mathematica. The material they received could be customised in a number of ways, for example depending on whether they had a suitable NAG Library on their machine or needed to access the software remotely, but this would be handled transparently. Documentation could be provided in the form of Mathematica notebooks. A particularly appealing aspect of this is that the interfaces could be created on demand, so that a user could gain access to our software with just a few mouse clicks.

In principal, of course, we could write such interfaces by hand but realistically this is not a viable proposition since it would certainly be error-prone and too costly to be an economic proposition. Our system is designed to work automatically, to take advantage of existing testing and verification regimes, and could in theory be extended to other bodies of software.

5 Describing Mathematical Services

What we have described so far are technologies for describing a mathematical object with a view to computing with it and for describing mathematical software with a view to accessing it via different environments. This existing technology takes us a long way towards being able to deploy mathematical services on the web in a practical and usable way. However the process of linking up servers and clients must, in practice, be done by hand. What we would like is a mechanism which allows us (a) to "publish" in some dynamic way the descriptions of the services which are currently available, and (b) to match a particular user's set of requirements to these services. This latter problem is particularly difficult since often we will be looking for a "best" rather than "exact" match.

We hope that frameworks for handling these two issues will emerge from the communities developing a semantic web. The languages being developed for the description of business services may also be used or adapted to address the needs of the mathematical community. Two languages in this area appear to be particularly relevant:

- UDDI [9] is a framework that aims to provide "a global, platform-independent, open framework to enable businesses to (1) discover each other, (2) define how they interact over the Internet, and (3) share information in a global registry that will more rapidly accelerate the global adoption of B2B eCommerce".
- WSDL [10] is an XML vocabulary for describing network services and operations, together with mechanisms for binding these to concrete network protocols such as SOAP and HTTP.

In addition to these, the Semantic Web activity is developing languages for defining Ontologies [7, 1]. The relationship between these and OpenMath Content Dictionaries, which in effect define an Ontology for mathematics, is yet to be formalised but may be important as a means of integrating mathematical service provision into the wider network of e-commerce and network services. In our context, the description of a service will include many features ranging from a specification of the mathematical task which it performs to information about whether problems submitted to it remain confidential, who is allowed to access it at what cost and so on. Much of this is common to all online services but parts of it are specific to mathematics.

We said earlier that we needed to be able to match a user's requirements to the available services. Some aspects of this will require detailed mathematical knowledge and possibly some initial analysis or computation. For example if a user wants to solve a differential equation then they may not know whether it is stiff or not. Choosing a non-stiff solver may eventually lead to a solution but it is definitely not a good strategy. An important class of mathematical service might therefore be an *advisor* which would assess the available services suitability to solving a particular problem. This implies that the services must advertise details of the methods they use, or at least their applicability. Such an advisor could learn from experience, and adjust the advice it gives accordingly. A number of "method selection" systems have already been implemented [2, 4] which rely on computer algebra systems to investigate underlying properties of a numerical problem. Mathematical software has also been used to provide "witnesses" for automatic theorem provers.

Another important service which might be combined with the advisor or be provided as a separate agent, would be a facility to explain how and why a particular answer was reached. This includes listing calls to services and explaining the reason why they were chosen, but would also need to include details of the computation and the approach or algorithm used which would be provided

by the service itself. As a simple example, consider the problem of computing $\int_{0.0}^{1.0} \frac{sin(x)}{x}\, dx$ which can also be denoted as the Sine Integral $Si(1.0)$. One possible approach is to recognise that this is in fact a well-known function and use either an online table to get the result or else compute an approximation via a Chebyshev expansion. Alternatively one might spot that the integrand had a singularity at a boundary and either use a method which could cope with this kind of situation, or recognise that it is a removable singularity and transform the integrand accordingly. Much of the above analysis requires some computation, and suggests that the distinction between an advisor and a service may not be very clear-cut. Which approaches was chosen probably would not matter as long as the result was something close to 0.9461. In more complicated cases a service might provide extra details of the computation to allow an independent check of the correctness (or even plausibility) of the result to be carried out. To handle the transfer and manipulation of this kind of information will require the development of special protocols and ontologies.

6 Conclusions

It is clear that we have many of the ingredients needed to deliver web-based mathematical services. Initial offerings in this area might appear to users to be similar to existing add-on packages (such as Matlab toolboxes) which simply give access to extra functionality inside a particular environment. However the underlying technology would be radically different and, critically from a practical point of view, add-ons in different packages could make use of much of the same underlying machinery.

Going beyond this simple model is harder, and requires the capability to handle errors and exceptional events, debug distributed computations, and help match a user's needs to available computational resources. In theory we could attempt to create protocols for all these tasks and then re-implement all our software to support them, but for the reasons mentioned in the introduction this is very undesirable and in any case will never happen in practice. The technologies being developed under the "semantic web" umbrella therefore appear to us to offer the best hope of achieving this and allowing existing legacy software to be deployed as web-based mathematical services.

Acknowledgements

The authors gratefully acknowledge the support of the European Union IST Programme through the OpenMath Project (24.969) and the OpenMath Thematic Network (IST-2000-28719).

References

[1] DAML+OIL (March 2001), March 2001. Available at `http://www.daml.org/2001/03/daml+oil-index`.

[2] Michael C. Dewar. Using Computer Algebra To Select Numerical Algorithms. In *Proceedings of ISSAC 1992*, pages 1–8. ACM, 1992.

[3] Mike Dewar, editor. *Special Issue on OpenMath*, volume 34 of *SIGSAM Bulletin*. June 2000.

[4] Brian Dupée and James Davenport. An Intelligent Interface to Numerical Routines. In *Design and Implementation of Symbolic Computation Systems (DISCO'96)*, pages 252–262. Springer-Verlag, 1996.

[5] B. Ford and F. Chatelin, editors. North Holland, 1987.

[6] Mathematical Markup Language (MathML) Version 2.0. W3C Working Draft 28 March 2000, March 2000. Available at `http://www.w3.org/TR/REC-MathML2/`.

[7] Resource Description Framework (RDF) Model and Syntax Specification. W3C Recommendation 22 February 1999, February 1999. Available at `http://www.w3.org/TR/REC-rdf-syntax`.

[8] The OpenMath Consortium. The OpenMath Standard. Technical report, September 2000. `http://www.nag.co.uk/projects/OpenMath/omstd`.

[9] Universal Description, Discovery and Integration, September 2000. Available at `http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf`.

[10] Web Services Description Language (WSDL) 1.1. W3C Note 15 March 2001, March 2001. Available at `http://www.w3.org/TR/2001/NOTE-wsdl-20010315`.