Записки научных
семинаров ПОМИ
Том 258, 1999 г.

Marc Gaëtano, Claude Huchet, and Winfried Neun

THE REALIZATION OF AN OPENMATH SERVER FOR REDUCE

1. Abstract

OpenMath is a standard for communicating mathematical objects between computer programs [3] in a multimedia computing environment. Such a standard in all its generality will revolutionize the workplace of a mathematician in the same way as the computer algebra tools did it in the last decade or the Internet in the last few years. To achieve this aim, processors have to pass the objects not in any string-type (a kind of denatured) form but they have to pass pointer into a mathematical description (called `Content Dictionaries` (short: CD) in the OpenMath project terminology) on which both sides agreed before. This is a requirement for being OpenMath compliant. The mappings between the OpenMath CD and the systems objects and data structures are called `phrasebook` and these have to be provided by the system developers. This paper describes the OpenMath approach shortly and summarizes related work done in ZIB recently. A prototype of the REDUCE OpenMath server has been presented at the OpenMath workshop in ZIB in January 1998 and will be available for public usage in the Internet in the near future.

2. Introduction

The workplace of a mathematician of today contains in most cases a collection of software tools running on a number of computer systems. These tools allow the users to perform tasks like formula manipulation, numerical computation, symbolic computation, visualization of mathematical objects or mathematical formula typesetting or editing. These tools should be able to exchange the objects under consideration easily, which very often is not the case. Moreover, through channels like Internet

* Home address: INRIA, Projet Safir, Sophia–Antipolis, France.
This paper was prepared while the authors were visiting the ZIB in 1997–1998. Claude Huchet is supported by the European ESPRIT project OpenMath. .

or electronic mail the workplaces of the mathematicians are connected, too. A simple question like: "How can I send this formula from my screen to my colleague, such that he is able to work with it without reformatting it" turns out to be not as simple as one might expect.

In order to solve this open problem a group of scientists formed the OpenMath working group in 1993. The discussion of the group is documented in the World Wide Web at URL http://www.openmath.org. It became very clear in the discussions that one needs to identify mathematical objects on both sides of a communication prior to sending them between processors. This requires a definition and a norm for the identification (which the OpenMath group has called a content dictionary (short CD)), and encodings for the transport layer.

TeX was an early answer to the question of encodings but it did not meet all the requirements one has, because it is mostly oriented towards typesetting. It does not allow a reconstruction of the mathematical content from the encoding. Another recent development is the draft for MathML [4], which allows one to add formulae to WWW pages and display formulae using a convenient browser. In this respect one might as well think of earlier systems connecting nodes running REDUCE on a massively parallel system using PVM [1] as transport vehicle as belonging to the same class of problems [2]. One might wonder whether the OpenMath system together with an efficient binary encoding using fast channels in the encoding library could be the method of choice for this type of communication among nodes.

In this paper we are going to discuss the OpenMath approach and its realization in REDUCE.

3. An introduction to OpenMath

The purpose of OpenMath is to provide a standard way of representing mathematical objects so that they can be exchanged between various programs. Examples of such programs include general purpose or specialized computer algebra systems, mathematical databases, mathematical typesetting systems, and so on. The exchange of mathematical data can be performed through various media such as files, mail, cut and paste and other interprocess communication means.

3.1. OpenMath architecture.

The overall architecture of an OpenMath application can be illustrated by the picture 3.2 which shows how two programs can exchange mathematical objects using the protocol. The first layer is the internal representation a given program uses to implement a mathematical object. The second layer is the corresponding OpenMath representation. The third layer is an encoding of the OpenMath object as a byte stream. This layered implementation corresponds to the way OpenMath is normally embedded into an application, although not all the three layers have to be actually implemented. For instance, the OpenMath objects may not be implemented as real objects in memory and the internal objects of the application could be read and written *on the fly*. The correspondence between an application-dependent object and the suitable OpenMath object is performed by a *phrasebook*, a piece of code that performs the translations. The definition of the correspondence is described in *content dictionaries* which simply describe which mathematical object any OpenMath object denotes.

3.2. OpenMath objects.

The OpenMath object can be viewed as an abstract term (or tree) whose leaves are the atomic objects: integers, floating-point numbers, symbols, variables and strings. Symbols carry the meaning of the object they describe. Each symbol belongs to a content dictionary and has a prescribed, defined meaning (described in the content dictionary). The nodes of the trees representing OpenMath objects are made of applications of a function (an OpenMath object) and a sequence of arguments (other OpenMath objects).

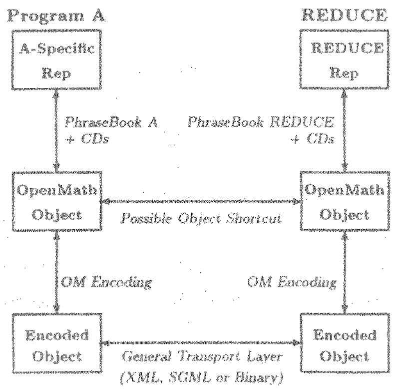

Fig. 1. The OpenMath Architecture.

3.3. Content dictionaries.

A content dictionary defines a set of symbols which are related to mathematical concepts and operations. They are written in a specific syntax and can be viewed as SGML documents (a DTD is available). For example, the constant π is defined in the BASIC content dictionary is the following way:

```
<CDDefinition>
  <Name> Pi </Name>
  <Description> The mathematical constant Pi
  </Description>
  <FunctorClass> Constant </FunctorClass>
  <CMP> Pi=3.141592653, ''10-digit approximation'' </CMP>
</CDDefinition>
```

The `Description` element gives as a comment the meaning of the symbol. The `FunctorClass` element gives some properties (such as the arity) useful for other programs. The `CMP` element contains a so-called "commented mathematical property". A `Signature` element is available to give the type of both the arguments and the result for functions, like in the exp function (still from BASIC):

```
<CDDefinition>
  <Name> exp </Name>
  <Description> The exponential function
  </Description>
  <FunctorClass> Unary, Function </FunctorClass>
  <Signature> real -> real </Signature>
  <Signature> symbolic -> symbolic </Signature>
</CDDefinition>
```

There are currently three *official* content dictionaries: META, the one which gives the definition of the CDs themselves, BASIC which gathers basic mathematical concepts and POLY dedicated to polynomial computations. Some other content dictionaries are under development, like LINALG for linear algebra or PROGRAMMING for basic programming to share programs and functions across different languages.

3.4. Encodings.

The current implementation of OpenMath supports three different encodings: a SGML-like encoding, an XML-like encoding and a binary encoding. The SGML and XML encodings are human readable and can be used to embed a piece of mathematics inside a SGML or XML document.

These encodings allow to cut and paste or to send through e-mail OpenMath objects. For example, the expression $\sin(x + 2k\pi)$ can be encoded using the SGML-encoding as:

```
<OMA>
  <OMS cd=''Basic'' name=''sin''>
  <OMA>
   <OMS cd=''Basic'' name=''+''>
   <OMV ''x''>
   <OMA>
    <OMS cd=''Basic'' name=''*''>
    2
    <OMV ''k''>
    <OMS cd=''Basic'' name=''Pi''>
   </OMA>
  </OMA>
</OMA>
```

The binary encoding is about 65% more compact than the SGML and XML one.

4. The OpenMath REDUCE server

A prototype version of a REDUCE OpenMath server based on the current specification of OpenMath has been implemented. This server is based on a C library for reading/writing OpenMath objects on various media such as files or sockets. This library has been developed at INRIA and implements an OpenMath Application Program Interface (API) available on UNIX machines. This library provides various functionalities for building and using devices, reading/writing OpenMath objects and handling interprocess communication. Apart from linking the C library into REDUCE, *phrasebooks* for the current CDs have been implemented in PSL (Portable Standard Lisp) [9], one of the underlying LISP systems of REDUCE ([7, 8]) for our implementation.

In fact, two versions of REDUCE have been developed as OpenMath-compliant systems. The first one is a CLIENT, that is a usual REDUCE with new features and functions to communicate with other OpenMath applications. The second one is the SERVER itself, that is an algebraic engine that can be launched by another process, a C program for example (in the following, we will show some examples where the server is launched by the OpenMath-REDUCE client). The main task of the server is to provide a toploop, which repeatingly performs the following steps: wait for an OpenMath object, decode it as a REDUCE expression, evaluate it and then send back the result of the evaluation express as an OpenMath object. The differences between the client and the server version are very small, in particular the phrasebooks are the same: they are sharing the same source files for the phrasebooks.

4.1. Implementation of the phrasebooks. Phrasebooks for the official CDs BASIC, POLY and INERT have been implemented and the first implementation of the not yet official CONTROL CD has been realized in this OpenMath-REDUCE version (see 4.2). They have been implemented in such a way that it is very easy to extend them for new CDs.

Phrasebooks for the current CDs have been implemented using appropriate macros to ease the translation of OpenMath objects into their REDUCE representation. For example, the following piece of code defines an entry for the symbol `minus` in the CD `Basic`:

```
(pb_defop !Basic minus
    (let ((args (read_args)))
     (cond ((null args)
       (pb_error '!Basic 'minus "no argument"))
      ((null (cdr args))
       (cons 'minus args))
      ((null (cddr args))
       (cons 'difference args))
      (t
       (pb_error '!Basic 'minus "too many arguments")))))
```

The macro +pb_defop+ associates a lambda which is called when the OpenMath symbol (its two first arguments) is read from the stream. In the previous example, `args` is the list of the arguments (read by +read_args+) of the application: if this list has one element, the `minus` REDUCE function is applied, else `difference` is applied. Some other details about a general way to implement this part of the phrasebooks for different systems can be found in [10].

At the opposite side, phrasebooks have been implemented to handle REDUCE internal objects and to send the corresponding OM object on the stream of the device. They use an appropriate set of PSL functions which associate each mathematical function implemented in REDUCE with the corresponding action on the stream. There are two different cases: either the REDUCE function corresponds exactly to an OpenMath symbol (this is the pbpr_sy function case, which takes the identifier of the REDUCE function as first argument and the corresponding OpenMath symbol as the second and third arguments), or more work must be done (this is the pbpr_op function which takes the identifier of the REDUCE mathematical function as first argument and the lambda doing the job as last argument).

```
(pbpr_sy 'quotient '!Basic 'over)
```

In this first example, the arguments are treated automatically by the general writing mechanism (that is the write_one function which appears for `recip` in the following) and only the correspondence between REDUCE function and OpenMath symbol is needed. Otherwise, every operations to write something on the stream is expressed (this is the task of the functions write_struct, write_one or write_symbol in the example below).

```
(pbpr_op 'recip
      '(lambda (x)
        (if (equal (length x) 2)
         (write_struct app
                (write_symbol 'over '!Basic)
                (write_one 1)
                (write_one (cadr x)))
         (err_code 'pbpr_recip "wrong number of arguments" x))))
```

As no exact mapping exists between REDUCE and OpenMath objects, new functions and objects have been defined at the user and/or internal level of REDUCE. For example, there is no basic construct in REDUCE to handle lambda expression although the lambda construct is defined in the `Basic` CD with the symbol `lambda`. New functions to built OpenMath objects are available at the user level (with name as `omapp`, `omsymbol`,..., some examples of their utilization can be found in the section 4.3).

4.2. Communication. The client and the server versions implement a very simple communication model between two OpenMath systems. The client sends an object and blocks until it gets an answer. The server is waiting for a query (an OpenMath object), when it gets one, it "evaluates" this object and return the OpenMath object corresponding to the result of this evaluation. In both cases, when an error is encountered then the object which is sent is (or contains) an OpenMath error object. The current discussion about different communication models for OpenMath and the CDs which go with, is reported at the following URL http://www.inria.fr/safir/OpenMath/hot/control.html.

In collaboration with Stéphane Dalmas from INRIA Sophia-Antipolis ([6]), an experimental CD (known as the CONTROL CD) has been written to formalize the new symbols necessary for this simple communication process and to describe the error symbols that are needed. Then, the phrasebook for this CD has been implemented in OpenMath-REDUCE. This CD contains, for example, the `commandOk` symbol which is the OpenMath symbol that must be send by the server as the answer of a query which has no return value. The phrasebook part for this symbol is very simple:

```
(pb_defsy !Control command!Ok t)
```

This means that this symbol is translated into the `t` (the *true* value) identifier of PSL.

4.3. Some examples. New instructions to read or write OpenMath objects and to communicate with other OpenMath applications, have been added to the the user interface of REDUCE.

```
Loading image file: /arsenal/claude/client.img
REDUCE/OPENMATH 1.0, 9-Jan-98
1: operator f
2: w := int(f(x),x);
w := int(f(x),x)
3: omwrite("test",w);
test
```

The `omwrite` function takes two arguments, the first one is a string (the name of a file) and the seconde one is a REDUCE usual object. The effect of this instruction is to write in the file the corresponding OpenMath object with the XML encoding. In this case, the file is:

```
< OMOBJ >
  < OMA >
    < OMS cd="Basic" name="integral"/ >
    < OMA >
      < OMS cd="Basic" name="lambda"/ >
      < OMV name="x"/ >
      < OMA >
        < OMV name="f"/ >
        < OMV name="x"/ >
      < /OMA >
    < /OMA >
    < OMV name="x"/ >
  < /OMA >
< /OMOBJ >
```

XML is the the default encoding of the `omwrite` function. Other possible encoding are SGML and binary. The encoding can be set by passing the new encoding as a third argument of the `omwrite` function.

The `omread` function is used to convert an OpenMath object stored in a file into a REDUCE object. This function does not require a default encoding, because the underlying C function is able to recognize the encoding from the first characters found in the file.

```
5: w1 := omread "test";
w1 := int(f(x),x)
```

The `omopen` instruction can launch an OpenMath server eventually on a remote computer and thus it creates a communication. The `omsend` instruction take as its last argument a REDUCE expresssion which is first evaluated and then the corresponding OpenMath object is send to the server. The return value of this function is the REDUCE translation of the OpenMath object returned by the server (in this special example the server name is `bin/server` and the machine name is `yorck`).

```
6: dev := omopen("yorck","bin/server","DISPLAY=yorck:0");
dev := *socket(yorck,bin/server)
```

```
7: w2 := omsend(dev,omapp(omsymbol(integral,!Basic),omlbda(y,sin(y)),x));
w2 := - cos (x)
```

In the example above, the REDUCE object which is send to the OpenMath server is the representation (in this system) of an OpenMath object. Here building an explicit OpenMath object can be useful for different reasons. The first one is to avoid the evaluation mechanism of REDUCE (as in this example). An second reason is that the object which is send can not always been expressed as a REDUCE expression, thus symbols for other CDs (than the available ones) can be used to built the object (with the help of the `omsymbol` function).

```
8: omclose(dev);
terminated
```

A connection is closed by the `omclose` function. First, the client sends the symbol `terminate` from the CONTROL CD and is waiting for the symbol `terminated` from the same CD as answer, then the server process can stop itself in a normal way and finally the physical communication is closed by the client.

5. Conclusion

We pointed out how we implemented a REDUCE-OpenMath Server in a three month effort and presented how far we got. The results were pretty encouraging. OpenMath as an emerging standard will be extraordinarily fruitful for the mathematical software community (and others, too), the more system designers commit themselves to offer their product in an OpenMath compliant version.

But, on the other hand, this represents some early steps in the process only. There is definitely a need for more content dictionaries to be created and fro more interesting applications to provide a OpenMath interface.

References

1. Oak Ridge, *National Laboratories: PVM 3 User's Guide and Reference Manual.* version 3.2, May 1993.
2. Winfried Neun, *REDUCE on a Massively Parallel System.* IMACS ACA 95 in Albuquerque, available on the WWW at URL: http://www.math.unm.edu/ACA-/1995/Proceedings /Sessions.html#parallel.
3. OpenMath Home Page, *currently available in the WWW at URL.* http://www.-openmath.org/.
4. World Wide Web Consortium, *MathML Overview Page, currently available in the WWW at URL.* http://www.w3c.org/Math.
5. Stéphane Dalmas, Marc Gaëtano and Stephen Watt, *An OpenMath.* v1.0 prototype, ISSAC, 1997.
6. Stéphane Dalmas, *OpenMath Version 1, special issue of the Euromath bulletin.* to appear, 1998.
7. Anthony C. Hearn and John P. Fitch, *REDUCE User's Manual 3.6; REDUCE 3.6 User's Guide for UNIX 3.6.* Konrad–Zuse–Zentrum, 1996.
8. Herbert Melenk and Winfried Neun, *PSL Users Manual.* ZIB Berlin, 1997, available on the WWW at URL: http://ftp.zib.de/pub/reduce/psl/manual.
9. Claude Huchet, *Writing OpenMath-compliant systems, special issue of the Euromath bulletin.* to appear, 1998.

Konrad–Zuse–Zentrum
für Informationstechnik Berlin (ZIB)
D–14195 Berlin-Dahlem
Federal Republic of Germany

Поступило 20 июня 1999 г.