An OpenMath 1.0 Implementation

Stéphane Dalmas Marc Gaëtano* Stephen Watt†

INRIA Sophia Antipolis, projet SAFIR

2004 route des Lucioles — BP 93

06902 Sophia-Antipolis CEDEX

France

stephane.dalmas@sophia.inria.fr
marc.gaetano@sophia.inria.fr
stephen.watt@sophia.inria.fr

Abstract

The first official version of the OpenMath specification was released in December. This paper presents the first implementation of this standard, in the form of a C library. To ensure a faithful realization, a second, independent implementation with the same API was built using Aldor ($A^\sharp$). We describe how the C library has been embedded in two main-stream computer algebra systems, Maple and Reduce, which can now communicate with each other and Aldor, and with specialized programs also linking the libraries. We discuss some of the problems encountered in developing the API, and the solutions we have chosen.

1 INTRODUCTION

The goal of OpenMath is to define a platform-independent standard for the representation of mathematical objects so that they may be exchanged in a meaningful way between various software tools.

This paper describes an implementation of the first official version of this specification. The libraries providing this implementation have been incorporated by principals of the Axiom/Aldor, Maple, and Reduce systems, allowing pre-release versions of these systems to inter-communicate.

This article presents OpenMath from a pragmatic point of view, relating the experience we have gained implementing two libraries and converting three applications. This perspective is influenced by our past experience in the design of mathematical communication and interoperability software [5] [6] [13].

OpenMath is still evolving. Feedback from this first implementation is necessary to ensure that it can successfully achieve its goals. By now, we feel it is certainly mature enough to be used, and to start to write OpenMath servers for existing applications. This experience will help focus future efforts.

The origin of OpenMath dates to 1992 and discussions among Maple developers as to how to best organize communication among mathematical software packages. At that time, Maple developers were emerging from the experience of developing bridges between Maple and Matlab, and Maple and Mathcad. Clearly there would be occasion to communicate with other software packages, and the process of developing special, proprietary mechanisms on each occasion was undesirable. An open standard for the communication of mathematical objects was needed. To this end, Prof. Gaston Gonnet organized a workshop at the ETH Zurich in December 1993, seeking the participation of potentially interested members of the computer algebra community.

Three years have passed since that first workshop, and in the intervening time many individuals have contributed time and effort to this goal of achieving an open standard. Steady progress has been made through a series of workshops, roughly one every six months.

A first detailed proposal was formulated by Stefan Vorkoetter [12]. These workshops then established the OpenMath Objectives Committee and the OpenMath Communications Committee. This served as a starting point for detailed electronic discussions, during which the committees formulated their advisory reports [2] [3] [9], and members produced a public report [1].

On December 15, 1996, the first official version of OpenMath was released [10], and is available through the OpenMath web site `http://www.openmath.org`. This release consists of a set of *content dictionaries* (CDs), covering basic mathematical concepts. This version takes on the committees' reports as a foundation, but differs in several significant respects.

Subsequent releases of OpenMath will include additional CDs, enriching the set of treated objects. (Also available is prototype software, for earlier experimental, versions of OpenMath.) The overall direction of the OpenMath effort has been governed by its Steering Committee, consisting of A. Cohen (Chair), G. Gonnet, M. Seppälä, R. Sutor and S. Watt.

During the period of OpenMath's evolution, a number of other communication schemes have developed or matured:

The *Multi Protocol*, developed by Gray, Kajler, Wang and others [7] [8], is an extensible protocol where user-supplied handlers allow efficient communication of mathematical objects, and is particularly well adapted to the communication of symbolic/numeric data.

The *Central Control* system provided a programmable

*also I3S, CNRS URA 1376,Université de Nice Sophia–Antipolis
†also IBM T.J. Watson Research Center, Yorktown Hts, NY (USA)

Reprinted from pp. 241-248 *Proc. International Symposium on Symbolic and Algebraic Computation (ISSAC 97)*, 21–23 July 1997, Kihei Hawaii, USA.

hub geared towards complex interactions between mathematical processes. Users provided interface classes describing the services of mathematical packages [5]. This was based on a simple underlying mathematical data communication scheme, *ASAP* [6].

The *MathLink* protocol has been used by Wolfram Research for user-interfaces and other programs to communicate with Mathematica computation servers[14]. Its use, however, requires proprietary software. Similarly *Maple* and *Axiom* have made use of proprietary protocols for communication between the mathematical engine and user interface processes.

The *MathBus* interchange format, developed by Zippel and others [15], is an interesting recent initiative focusing on a small, low-level core of structures into which representations used by a wide variety of languages can be mapped.

The *HTML Math* Working Group has as its mandate to define a format for representing mathematical formulae in HTML [11].

Each of these efforts is somewhat related to OpenMath, but are largely orthogonal to OpenMath's key issue: *How to convey specific mathematical objects and commands between algebra systems so that their meanings are preserved.* To a first approximation, the low-level interchange mechanisms (ASAP, Mathlink, MathBus) do not concern themselves with high-level mathematical meaning. The interfaces oriented toward programming (Central Control, MP, MathBus), do not concern themselves with what, specifically, is transmitted. The work in HTML Math is primarily oriented toward rendering.

Because of this orthogonality, many of the participants in the works cited above are also are engaged in the OpenMath definition. For example, the HTML Working Group has about a 20% overlap with OpenMath, and is seriously considering using OpenMath annotations to convey mathematical meaning in those cases where it is desired.

This article is organized as follows: In section 1 we have presented the history of OpenMath and its relation to other work. Section 2 describes the salient technical aspects of the first release of OpenMath, including *"content dictionaries"*, *"phrase books"*, and *"encodings"*. Section 3 presents our C-language applications programming interface (API) for building and communicating OpenMath objects. Section 4 provides a somewhat detailed example of how our library has been incorporated into the Maple computer algebra system. Section 5 describes an Aldor library realizing the same API and inter-operable with the C library. The article closes with some general conclusions and acknowledgements.

2 OPENMATH

The goal of OpenMath is to define a standard way of representing mathematical objects so that they can be exchanged between various software tools. Examples of such tools are general purpose or specialized computer algebra systems, document preparation systems, Web browsers displaying formula, equation editors, databases containing mathematical information and so on. The exchange can take place through various media such as regular files, electronic mail, cut and paste and other interprocess communication means.

The motivation for defining OpenMath is well understood. A real need exists: people would like to write programs (for example, numerical programs, for efficiency reasons) that can be run from computer algebra systems, use specialized programs (usually not available as servers) from their favorite general purpose system or use symbolic computation facilities (or some other existing mathematically oriented program) from their own applications.

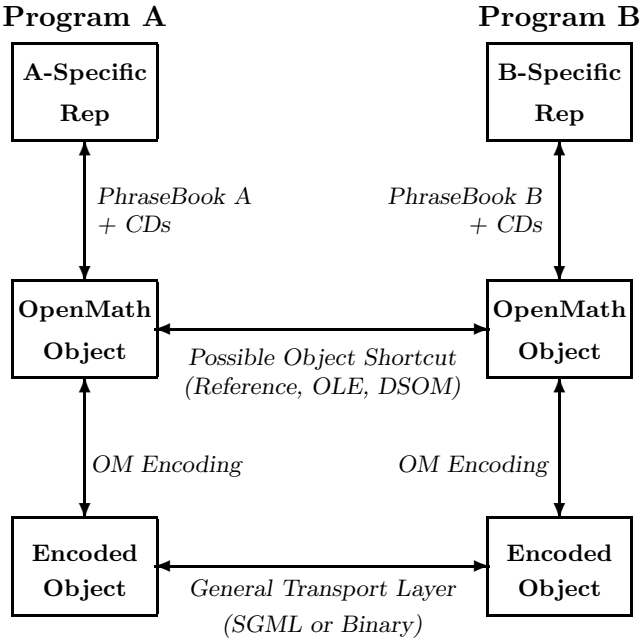

Figure 1: The OpenMath Architecture

The user would like to be able to exchange examples or results via e-mail or try the same problem, copied from a Web page, in Axiom, Maple or Mathematica. A common semantically rich format to represent mathematical objects in texts would allow powerful search engines to be built to explore databases of mathematical articles.

2.1 Representing mathematical objects

We distinguish three layers of representations for a mathematical object. The first is a private (internal) representation to a given program. The second is its representation as an OpenMath object. The third is a representation of the corresponding OpenMath object as a byte stream that is used as the external form (the low-level form that is actually exchanged). The hierarchy of these layers corresponds to the way OpenMath is normally integrated into an application. The application uses its internal representation for an object. This can be converted to an OpenMath object, and then saved or transmitted as a byte stream.

The correspondence between an object in an application (in its application-dependent representation) and an OpenMath object is done via a *phrase book*. A phrase book is a piece of software that performs the translations. How to establish the correspondence is described in *content dictionaries*. A content dictionary maps the semantic level to the OpenMath level and describes which mathematical object the corresponding OpenMath object denotes. This overall organization is pictured in Figure 2.

The OpenMath object can be considered abstract: An application does not need to construct OpenMath objects. It could, for example, simply output encodings that are pre-

defined implementations of such objects. An intermediate "data-structure" level can also be considered. This level would describe an implementation of OpenMath objects inside an application.

2.2 OpenMath objects

An OpenMath object can be viewed as a term or a tree. The leaves are the basic and atomic objects: integers (signed, of unbounded length), floating-point numbers, symbols and strings. Although one might argue that floating-point numbers are not true mathematical objects, their use in mathematical applications is often necessary, and OpenMath also aims to be useful for numerical applications. Two additional classes of basic objects are supported by the prototype: variables and byte arrays (used to exchange arbitrary binary data).

For strings and floating-point numbers, there are some implementation choices to be made regarding the character set and the formats supported. Strings contain Unicode characters. Floating-point numbers can be both single and double precision and follow IEEE formats.

Symbols are at the heart of OpenMath. Each symbol has a name and is a member of a content dictionary. Each symbol has a prescribed, defined meaning. This meaning is given (informally) in its home content dictionary.

The nodes of the trees representing OpenMath objects are made of applications (and Error objects, in our prototype). An application is made of a function (an OpenMath object) and a sequence of arguments (other OpenMath objects). Errors are made of a symbol and a sequence of arguments (arbitrary OpenMath expressions).

Any OpenMath object may be attributed with a sequence of pairs, each pair consisting of an attribute name (a symbol) and an attached value (an OpenMath object).

2.3 Content dictionaries

A content dictionary defines a set of related concepts and operations as a set of symbols. It can also define one or more representation for some objects (such as multivariate polynomials).

Content dictionaries are written in a specific syntax and can be viewed as SGML documents (a DTD is available). It is possible to translate this form to OpenMath, making a content dictionary an OpenMath object. We have written a program (`cd2om`) that performs this translation as part of our prototype. The structure of a content dictionary is specified by a special content dictionary, called *Meta*.

Some information is attached to the symbols in a dictionary. There is a `Description` element, giving, in words, the meaning of the symbol. A `FunctorClass` element gives properties, such as the arity, for use by other programs. A `Signature` element is used to give the type of the arguments and the result (for functions), using a little type language. Further information on the entries of a dictionary can be found in the *Meta* dictionary itself. As a first example, here is how the boolean value *true* is defined in the *Basic* dictionary:

```
<CDDefinition>
   <Name> true </Name>
   <Description>
   The boolean value true
   </Description>
   <FunctorClass> Constant </FunctorClass>
</CDDefinition>
```

This is the definition of the multiplication operator in the same dictionary:

```
<CDDefinition>
  <Name> * </Name>
  <Description> The multiplication operator of any
     commutative ring </Description>
  <FunctorClass> Binary, Operator </FunctorClass>
  <CDAttributes> Associative, Commutative
  </CDAttributes>
  <Signature> (complex complex) -> complex
  </Signature>
  <Signature> (real real) -> real
  </Signature>
  <Signature> (rational rational) -> rational
  </Signature>
  <Signature> (integer integer) -> integer
  </Signature>
  <Signature> (symbolic symbolic) -> symbolic
   </Signature>
  <CMP> a*b=b*a, commutativity </CMP>
  <CMP> a*(b*c)=(a*b)*c, associativity </CMP>
  <CMP> a*1=1*a=a, identity </CMP>
  <CMP> a*0=0*a=0 </CMP>
</CDDefinition>
```

The `CMP` element contains a "commented mathematical property" (in the form of an equation or formula and an associated description).

Apart from *Basic*, two other dictionaries have been written, one for handling multivariate polynomials, *Poly*, and one for linear algebra. The dictionary for polynomials is quite different from *Basic*. Although it basically defines a set of concepts related to polynomials (such as degree, factorization, resultant...), there are two noticeable points: a certain emphasis on representation issues (including structural constraints on some OpenMath objects) and an attempt to specify some computational behaviour for OpenMath applications that handle (part of) this dictionary.

One of the interests of OpenMath is to enable the use of specialized servers. It is important to promote the writing of OpenMath-compliant servers by placing as few constraints as possible on the programmers of these packages. The *Poly* dictionary has been designed with the idea that it could be simple to use for a server dealing only with polynomial computations. Hence we have defined a particular representation for polynomials (distributed with dense monomials). This representation is rather abstract as it does not introduce names for variables. It explicitly contains the polynomial ring as the set of the coefficients and the number of variables. In our experience, this information is necessary for many specialized servers.

It is not always easy to express constraints on the structure of OpenMath objects made from the symbols of a dictionary. One of the main reasons is that a symbol such as `gcd` is meant to denote the GCD of a set of polynomials, no

matter how the polynomials are represented. Such a function should thus accept both "symbolic" arguments (a list of symbolic objects meant to be polynomials) and the polynomials in the specific representation defined in *Poly*. Another solution would be to have one `gcd` for one (or several) particular representation and another `gcd` to express the general notion of polynomial GCD. The solution we have chosen is, however, more in the spirit of *Basic* and the opinions expressed during OpenMath meetings.

One question we have not answered is whether it should be possible to have "symbolic" objects inside certain constructors (e.g. a power which is not an OpenMath integer as an argument of a monomial constructor, or a symbolic ring (a variable) as an argument of the constructor for polynomials). We explicitly forbid this in the first version of *Poly*.

2.4 Specifying the meaning of the symbols and the behavior of OpenMath applications

The purpose of OpenMath is to define representations for mathematical objects. It is not intended to be used to completely specify the behavior of applications using mathematical objects. In fact, it would be beside the point to exactly specify the behaviour of any OpenMath, for at least two reasons:

- an OpenMath object is intended to represent a mathematical object and thus the same OpenMath object could be sent to many applications, e.g. a typesetter, and a symbolic computation system,
- even when dealing with programs that compute, exact specifications could be impractical or too constraining for a given system to become OpenMath compliant.

On the other hand, one of the goals of OpenMath is that a program needing to perform a mathematical operation should be able to dispatch the request to one of several systems. For example, to factor an integer a program could use Maple, Axiom or Pari to do the job. This is possible only if all severs computing integer factorizations answer in a similar way. We should therefore not hesitate to specify what OpenMath applications (the computing ones) should return as results. Compliance in this sense is simple enough, and is obviously very useful. This kind of specification can be made with a dictionary using particular symbols to encapsulate the results. An example would be a symbol `factored` to express the result of a factorization or `GroebnerBasis` to describe the result of a Gröbner basis computation.

Concretely, a general compliance rule for the *Poly* CD can be stated as: for an OpenMath application to recognize this dictionary, it must implement some of the operations to provide results using *Poly* constructors.

This means that if the OpenMath version of a computer algebra system claims to implement polynomial factorization, another application can send it an OpenMath object as described in the comment associated to the `factor` symbol and the result will be return as defined, *i.e.* as the `factored` symbol application whose arguments are described in the corresponding entry of the dictionary.

3 A C INTERFACE

We have designed a first C Application Programming Interface (API) for OpenMath and implemented it as a library. At present the library has been tested only on Unix-based machines, even though an earlier version has been deployed on Mircosoft Windows.

The current API can be partitioned in three parts: The first part deals with OpenMath objects, content dictionaries and devices (described below). The second part contains a set of functions to create various kind of devices. The third part contains functions to implement interprocess communication.

The first part can be further divided: There is a first set of functions to deal with symbols and content dictionaries. A second set deals with abstract *devices*, from which OpenMath objects are read and written. The operations on devices are such that they may be realized with files, sockets or shared objects. A third set of operations build OpenMath expressions in devices, and read OpenMath expressions from devices.

These three sets could be complemented with a fourth for manipulating OpenMath objects directly in an application. However, it is not clear at present that there is a real need for this set.

The current devices are composed as two-layered objects. The top layer is for encoding and the bottom layer is for I/O. This I/O layer abstracts the manner in which data corresponding to encodings are read and written. The encoding layer uses the facilities provided by the I/O layer for low-level input and output.

In the current library, a device provides access to the two structures corresponding to its two layers, allowing the programmer to develop new I/O possibilities and new encodings. This last possibility is clearly temporary, to support easy experimentation.

3.1 Building OpenMath objects

The most important function of an OpenMath library is to provide operations to build OpenMath objects. Our API implements a model similar to that of *MathLink* [14] or ASAP [6], where expressions are built inside a device. We give an example showing how to output (build) the expression `x + y`. First, we would obtain the `+` as a symbol from the *Basic* dictionary. There are several ways to do this. The simplest is to define the content dictionary and construct the symbol in it:

```
OMCD_t *Basic;
OMsymbol_t *plus;
Basic = OMgetCD("Basic");
plus = OMmakeSymbol(Basic, "+");
```

Then the expression can be "built" in the device `dev` by the following sequence of operations:

```
OMbeginObject(dev);
OMputApp(dev);
  OMputSymbol(dev, plus);
  OMputVar(dev, "x");
  OMputVar(dev, "y");
OMputEndApp(dev);
OMendObject(dev);
```

Unlike ASAP and MathLink, the API implements a bracketed model for compound expressions: One starts with a "begin" call. Then the subexpressions are given (in the case of an application the first argument is the function). Finally, an "end" call closes the construct. The building of a

compound object in a device always begins with a call to `OMbeginObject` and ends with a call to `OMendObject`.

3.2 Building devices

Devices are built with the `OMmakeDevice`. This function's first argument specifies an encoding by a symbolic constant, and the second argument is a pointer to an `OMIOstruct_t` object. These objects are normally produced by other functions of the API. For example, to create a device that reads from the standard input on uses:

```
dev = OMmakeDevice(OM_ENCOD_SGML, OMIOFile(stdin));
```

The `OMIOFile` function is used to create the I/O structure that reads from `stdin`.

Two different encodings are available. The first, `OM_ENCOD_SGML`, is a SGML-like encoding that uses only "printable" ASCII characters. It is human-readable, editable and suitable for transmission via e-mail and can be included in documents. `OM_ENCOD_BIN` is the "binary" encoding, which is more compact (about 65%) and faster to encode and decode. It is therefore the encoding of choice for interprocess communication. Note that there was no real effort to produce an encoding as compact as possible. Using general-purpose compression algorithms on binary encoded objects can reduce the size of the message by a factor of three. For compact transmission, one would compose the I/O layer with a compression layer and retain the advantages of straightforward code.

The library provides several functions to construct I/O structures, such as `OMIOFile`s, to use `FILE` objects from the standard C library, or `OMIOfd` to use UNIX file descriptors. The `OMIOstringIn` and `OMIOstringOut` functions can be used to make devices that input and output to strings, and can be useful, for example, to implement cut-and-paste. It is possible to build an "internal device" to keep the object in memory (shared memory).

3.3 Interprocess communication

The communication layer is built on the device layer, and uses *connections* as its basic abstraction. A connection is a set of devices that can be used to send and receive OpenMath objects.

Connections are described by the `OMconn_t` type, An `OMconn_t` is a structure with two user-accessible fields `in` and `out`. `in` is a pointer to a device to be used for input. `out` is pointer to a device to be used for output.

The I/O structure in a device provides all the necessary interface functionality to use a broad range of transmission or communication means (e.g. distributed object protocols). The current library directly provides some facilities based on TCP for handling connection-oriented communication.

The TCP-based functions allow binding and connections at particular IP addresses (to implement or connect to servers running at some specified addresses) as well as a way to start a given server (establishing the connection). The C prototypes for these functions are

```
OMstatus_t OMlaunch (OMconn_t *con,
             char *mach, char *cmd);

OMstatus_t OMlaunchEnv (OMconn_t *con,
             char *mach, char *cmd, char *env);

OMstatus_t OMserveClient (OMconn_t *conn);
```

where `mach` is a machine name, `cmd` is a command to run the server and `env` is an environment (a sequence of environment variables with their values). `OMserveClient` is the function that has to be called in the server (when it is launched by its client) to establish this side of the connection.

These functions use the remote shell (`rsh`) mechanism to run the commands. The addresses are chosen dynamically and passed via environment variables to the remote processes.

For applications that run on the same (UNIX) machine, a more efficient interprocess mechanism can be used (UNIX domain sockets). `OMlaunch` and `OMlaunchEnv` transparently do this if the remote machine is `localhost`.

3.4 Peculiarities of interprocess communication

It has been recognized in the OpenMath meetings that a broad range of problems arise from general multiprocess configurations. For concreteness, the initial attention has focussed on the case where two processes exchange data.

Going from a simple model (where objects are read from and written to persistent storage) to real interprocess communication, can introduce several technical problems with protocols, negotiations and urgent messages. Although this can be legitimately considered as outside the scope of the OpenMath definition, we need to cope with this in an OpenMath implementation. Otherwise, the specification may be useless for certain classes of applications and some users may be forsed to implement ad-hoc solutions, rendering interoperability very difficult.

When two processes use the same internal format for machine integers or floating-point numbers, it is natural to want the ability to use that format, i.e. to negotiate the whole encoding or part of the encoding.

Especially for interactive applications, such as a graphical interface talking with a computation engine using OpenMath, it is necessary to support the possibility of interrupting a computation or, more generally, to support out-of-band messages (urgent messages that are not part of the normal communication, usually occurring asynchronously). An example where this kind of facility could be useful would be a message sent during a calculation to query how much processor time had been used.

One difficult point is the handling of interrupts which occur while a response is being generated, especially when the response is large or sent as it is being computed. In this case, it is necessary that the sender be able to abort the transmission of an object (because the receiver cannot wait until it is finished). A special "premature end of object" tag can be sent to this effect. To overcome the problem of a large item, it is sufficient to internally "chop" data into large blocks.

There is no real problem for handling negotiations. It can be done transparently (from a user point of view) in the functions that set up the connections.

Handling interruptions and out-of-band data is more problematic and system-dependent. Depending on the way the application is organized and the operating systems the two processes are running on, various methods can be used for interruptions, including explicit polling on some conditions and handling an asynchronous event such as a signal either explicitly sent or received as the result of setting an urgent condition on some object (such as a socket descriptor).

To avoid resorting to the lowest common denominator (and encouraging incompatible ad-hoc solutions), the features that are needed can be specified in `OMmakeConn` and checked by the functions that can perform the connection. If the connection cannot be made with the required feature, it will fail. This is the reason why creating a connection is separate from establishing the connection.

4 OPENMATH AND MAPLE

Our C library has been linked with a development version of Maple provided by Waterloo Maple Incorporated. In this OpenMath version of Maple, all the functions in the C library can be accessed through Maple procedures. On top of that, we have written some Maple code to implement the phrase book (translations between OpenMath objects and Maple objects) and some useful functions (such as functions that read/write OpenMath objects to files).

This version can communicate with (and run) OpenMath applications. For example, it can compute a Gröbner basis through the Aldor server (see 5.2). Here is the (somewhat simplified) source code for the `animGrobner` procedure that uses the Gröbner server:

```
# For visual feedback, run the server in an "xterm"
ANIMSERVER :=
 "xterm -e /u/kama/safir/gaetano/ALDOR/ANIM/anim1";

animGrobner := proc(display, machine, l)
  local l, display, conn;
  global ANIMSERVER, keepPolyVarName;

  conn := OMmakeConn(5000);
  if not
    OMlaunchEnv(conn, machine, ANIMSERVER,
                "DISPLAY=" . display) then
    ERROR("cannot connect to server");
  fi;
  # Now the connection is established and
  # we can send the polynomials as a list
  OMwrite(OMconnOut(conn), Maple2DMPList(l));

  # to avoid renaming the variables
  keepPolyVarName := true;
  # convert the result of the animation program
  # back to Maple.
  RETURN (OM2Maple(OMread(OMconnIn(conn))));
end:
```

This version of Maple can also be called by another application and can thus be used as an OpenMath Maple server. This feature was implemented at the Maple level by reading a startup file which calls the `OMserveClient` function and `OMserve` in a loop. This last function can be defined as:

```
OMserve := proc(conn)
  local req, res;
  req := OMread(OMconnIn(conn));
  res := Maple2OM(OM2Maple(req));
  OMwrite(OMconnOut(conn), res);
end:
```

Here are a few numbers that give an idea of the effort involved in making Maple an OpenMath application. The basic interface to the C library and Maple functions that read/write OpenMath objects are 300 lines of Maple code. The functions that support the Maple phrase book are another 500 lines of Maple code (this includes support of both the *Basic* and *Poly* dictionaries). The most cumbersome part in implementing the phrase book was writing the code to check that expressions have the correct form, and the code to deal with errors.

4.1 A Maple phrase book

The Maple phrase book is implemented by two functions. Each function performs the translation in a top-down manner using a table. In many cases, the correspondence is trivial and the table records a simple mapping between OpenMath symbols and Maple symbols. The `OMregisterSimple` function is used to declare the mapping:

```
OMregisterSimple("Basic", "pi", Pi);
OMregisterSimple("Basic", "exp", exp);
```

When the mapping is more complicated the tables record functions that are called to translate OpenMath or Maple expressions. In the *Basic* dictionary, the integration and differentiation operators use a lambda abstraction to represent functions whereas Maple uses plain algebraic expressions. The translation for derivatives is declared as

```
OMregisterOM("Basic", "diff", OMdiffToMaple);
OMregisterMaple("diff", OMMapleToDiff);
OMregisterMaple("Diff", OMMapleToDiff);
```

where `OMdiffToMaple` and `OMMapleToDiff` are some suitable functions that receive the whole (untranslated) arguments of the corresponding operators.

Implementing the phrase book for *Basic* is essentially trivial. Implementing the phrase book for the *Poly* dictionary is a little bit more complicated due to the fact that Maple does not use a canonical representation for polynomials. In this case, specific functions have been written to explicitly convert Maple expressions to polynomials using the constructors defined in this dictionary. The inverse mapping is realized by attaching suitable functions to the constructors (such as `DMP` and `DMPL`).

5 OPENMATH AND ALDOR

From a long term perspective, APIs for OpenMath should be developed for various programming languages. For many other languages, it will be sufficient to link the C library and provide cover functions. This will not always be possible, however, so it is important to verify that two independent implementations can interact properly.

A natural choice of another language for experimenting with OpenMath principles, was Aldor ($A^\sharp$) — the Axiom extension language [13]. Aldor is a strongly typed language based on category and domain constructors. In this context,

every mathematical object has a unique type. This differs from systems such as Maple or Reduce, and trying to make these systems exchange mathematical objects is a typical example of what OpenMath aims to achieve. (We note that it would have been possible to use the C library directly from Aldor, but this would not have provided the verification we sought.)

A first Aldor API for OpenMath has been implemented as a set of domains and has been tested on UNIX machines. An Aldor server, based on this library performing animated Gröbner basis computation, has been developed and tested with Maple and Reduce.

5.1 The Aldor API

The Aldor library is based upon the same structure as the C library. The layered implementation of devices in the C API is made explicit in Aldor by using parameterized categories and domains. The I/O layer is abstracted by the `OMioCat` category which exports the minimal set of functions needed for reading/writing the data exchanged by OpenMath compliant applications. The encoding layer is abstracted by the category `OMcodingCat`, parameterized by a given I/O layer. It is then possible to combine any encoding (a *domain* of category `OMcodingCat`) with any I/O layer (a *domain* of category `OMioCat`).

A domain implementing the I/O layer must export basic functions for low-level input and output of characters:

```
define OMioCat: Category == with {
  CHAR ==> Character;
  OMS  ==> OMstatus;

  flush:     % -> OMS;
  putchar:   (Char, %) -> OMS;
  print:     (String, %) -> OMS;
  getchar:   % -> Union(stat: OMS, char:Char);
  lookahead: % -> Union(stat: OMS, char:Char);
}
```

As the current version of the Aldor compiler did not support any exception handling mechanism[1], a function which writes to or reads from an I/O object must return an `Union` type to handle a possible failure. The `OMstatus` type expresses the relevant information:

```
OMstatus: BasicType with {
  success:        %;
  syntax__error:  %;
  system__error:  %;
  unknown__error: %;
} == add {
  ....
}
```

Notice that an I/O error is different from an OpenMath error object, which is a possible answer returned by an OpenMath application. Similarly to the C API the OpenMath expressions are "built" inside the device using a bracketed model for compound expressions:

[1]Note in proof: Exceptions have been added in the interim, and are now in alpha test.

```
define OMcodingCat(IO: OMioCat): Category == with {
  OMS  ==> OMstatus;

  integer:     (IO, Integer) -> OMS;
  float:       (IO, Float) -> OMS;
  string:      (IO, String) -> OMS;
  ...

  beginApply:  IO -> OMS;
  endApply:    IO -> OMS;
  beginAttrib: IO -> OMS;
  endAttrib:   IO -> OMS;
  ...

  gettype:     IO -> Union(stat: OMS, typ: OMtype);
  gettag:      IO -> OMS;

  integer:     IO -> Union(stat: OMS, int: Integer);
  float:       IO -> Union(stat: OMS, flo: Float);
  string:      IO -> Union(stat: OMS, str: String);
  ...
}
```

The functions exported by a domain, typed by the category `OMcodingCat` (i.e *an encoding*), input and output an OpenMath encoding directly on the I/O and may avoid representing OpenMath objects in memory. Nevertheless, the data-structure level has been implemented through the domain `OMexpr`, and OpenMath objects can be built and stored in an Aldor program. For now, two I/O-domains and two encoding-domains have been implemented. The `OMfile` and `OMsocket` domains respectively implements a file-based and a socket-based I/O layer. The `OMbinary` and `OMsgml` domains respectively implements a binary and a SGML-like encoding of OpenMath objects. Given an I/O-domain as parameter, the `OMbinary` (or `OMsgml`) domain constructor returns an OpenMath *device*. For example, the line

```
import from OMsgml(OMsocket);
```

provides an Aldor program with all the functions to read and write OpenMath objects encoded in an SGML-like format on a socket-based device.

5.2 An Aldor server

If the use of OpenMath becomes widespread, it is expected that a number of existing mathematical programs will be converted to OpenMath compliant servers so as to be broadly used by other programs. To experiment with the OpenMath Aldor library in this spirit, we converted an existing Aldor program to an OpenMath server. To understand an *ab initio* conversion, we selected a program *not* written by any of the authors of this article. This program animates the computation of a Gröbner basis of a set of polynomials over the rationals for a fixed ordering [4]. Given the *phrase book* to translate back and forth between a polynomial and the corresponding OpenMath object, the transformation is rather trivial and consists mainly in replacing standard input/output calls by those provided by the OpenMath library. The animation server understands objects from the *Poly* content dictionary, and can be used by either Maple, Reduce or another Aldor program.

6 CONCLUSION

Concrete implementations have been an important step in the evolution of OpenMath. The initial prototypes developed by Stefan Vorkoetter and John Abbott have been important first steps, and we feel that the present work has been a useful next stage.

Cooperation of computer-algebra system principals has been a key ingredient in our being able to demonstrate intermediate versions of this work at OpenMath workshops over the past year. From this we have received immediate feedback, allowing us to adapt our approach in response. Consequently, at the OpenMath workshop in Dublin, parts of our demonstration version were sufficiently mature to be taken almost directly into the first official OpenMath release.

At this point OpenMath is ready for external use. With the CDs defined in OpenMath version 1.0, it is possible for two mathematical programs to exchange free expressions of a general class, as well as semantically rich multivariate polynomials. This is immediately useful, *e.g.*, in circumstances where the same test suite must be used for programs in different computer algebra environments, as in the polynomial test suite for FRISCO (ESPRIT IV LTR 21.024).

The libraries described in this paper are being placed on the INRIA FTP server, as well as `www.openmath.org`, so individuals will not have to program from scratch to make use of OpenMath. In fact, as the transport layer for mathematical objects may evolve somewhat over time, those wishing to insulate themselves from this change could program using this API and rely on the library to track any changes.

Much time has been spent in the OpenMath meetings on the detailed implementation aspects of this protocol. It seems that now the most challenging and important work is in the definition of CDs which are at once well-defined and useful for additional areas of mathematics.

7 ACKNOWLEDGMENTS

OpenMath is the joint work of many people. Hence, part of the material we have presented here is the direct result of discussions held at the OpenMath workshops and on mailing lists. This article does not pretend to give an official view of OpenMath. Nevertheless, we have done our best in trying to express what has been agreed upon by the OpenMath consortium.

The OpenMath libraries presented in this paper would be much less interesting if they had not benefited from several rounds of integration into real computer algebra systems:

We thank NAG Limited, and in particular Peter Broadbery, for providing up-to-date versions of Aldor and responding to our exception-handling needs.

We thank Waterloo Maple Incorporated, and Stefan Vorkoetter in particular, for invaluable assistance in providing a Windows port and in linking our libraries into various versions of Maple.

We thank Winfried Neun of ZIB (Berlin) for having incorporated our libraries in various versions of Reduce, and for providing the Reduce phrase books (under extreme time pressure).

We thank Yannis Chicha of the University of Nice, for building a version of his Aldor functional animation engine incorporating our libraries, and Manuel Bronstein for providing a version of Bernina as a test case.

References

[1] ABBOT, J., DIAZ, A., AND SUTOR, R. S. A report on OpenMath. *SIGSAM bulletin 30*, 1 (1996), 21–24.

[2] ABBOTT, J. OpenMath design committee report: Version 23, Dec. 1996. `http://www.w3.org/OpenMath/History/reports/design-report.ps.gz`.

[3] ABBOTT, J., VAN LEEUWEN, A., AND STROTMANN, A. Objectives of OpenMath. Technical Report 12, RIACA, 1996.

[4] CHICHA, Y. Animation de programmes fonctionnels. Master's thesis, Université de Nice, 1996.

[5] DALMAS, S., AND GAËTANO, M. Making systems communicate and cooperate: the central control approach. In *Design and Implementation of Symbolic Computation Systems*, J. Calmet and C. Limongelli, Eds., no. 1128 in LNCS. Springer Verlag, 1996, pp. 308–319.

[6] DALMAS, S., GAËTANO, M., AND SAUSSE, A. ASAP : A Protocol for Symbolic Computation Systems. Tech. Rep. 162, Institut National de Recherche en Informatique et en Automatique, Mar. 1994.

[7] GRAY, S., KAJLER, N., AND WANG, P. MP: A protocol for the efficient exchange of mathematical expressions. In *Proc. ISSAC 94* (1994), ACM Press, pp. 330–335.

[8] GRAY, S., KAJLER, N., AND WANG, P. S. Pluggability issues in the multi protocol. In *Design and Implementation of Symbolic Computation Systems*, J. Calmet and C. Limongelli, Eds., no. 1128 in LNCS. Springer Verlag, 1996, pp. 343–356.

[9] THE OPENMATH COMMUNICATIONS COMMITTEE. Openmath communications committee report. Technical Report 11, RIACA, 1996.

[10] THE OPENMATH STEERING COMMITTEE. Openmath version 1.0 released, Dec. 1996. `http://www.openmath.org`.

[11] THE WORLD WIDE WEB CONSORTIUM. HTML math working group, 1997. `http://www.w3.org/pub/WWW/MarkUp/Math`.

[12] VORKOETTER, S. Proposed OpenMath specification: Draft version 1.1, July 1995. `http://www.w3.org/OpenMath/History/reports/prototype0-spec.ps.gz`.

[13] WATT, S. M., BROADBERY, P. A., DOOLEY, S. S., IGLIO, P., MORRISON, S. C., STEINBACH, J. M., AND SUTOR, R. S. A first report on the $A\sharp$ compiler. In *Proceedings of ISSAC'94* (1994), ACM press, pp. 25–31.

[14] WOLFRAM RESEARCH. *Mathlink Reference Guide (version 2.2)*, 1993.

[15] ZIPPEL, R. Mathbus, Feb. 1997. `http://www2.cs.cornell.edu/Simlab/papers/mathbus/mathTerm.htm`.