OpenMath in SCIEnce: SCSCP and POPCORN

Peter Horn[1] and Dan Roozemond[2]

[1] Universität Kassel, Heinrich Plett Straße 40, 34132 Kassel,
horn@math.uni-kassel.de,
http://www.mathematik.uni-kassel.de/ hornp
[2] Technical Universiteit Eindhoven, Den Dolech 2, Postbus 513, 5600 MB Eindhoven,
d.a.roozemond@tue.nl,
http://www.win.tue.nl/ droozemo

Abstract. In this short communication we want to give an overview of how OpenMath is used in the European project "SCIEnce" [12]. The main aim of this project is to allow unified communication between different computer algebra systems (CASes) or different instances of one CAS. This may involve one or more computers, clusters, and even grids.

The main topics are the use of OpenMath to marshal mathematical objects for transport between different CASes, an alternative textual OpenMath representation more suitable for human reading and writing, and finally the publicly released Java Library developed for the project.

1 Marshaling Mathematics in SCSCP

When designing a uniform communication interface for Computer Algebra Systems, the first problem that needs to be solved is how to transport the mathematical objects from one system to another. Here, the obvious choice for us was OpenMath [6], since it is a widely used standard with a long history of assisting communication between CASes [1,2]. In this section we briefly comment on the problems faced and the choices made.

To simplify the communication between the various CASes, we have developed a protocol called "Symbolic Computation Software Composability Protocol", abbreviated SCSCP [9,13]. This protocol does not only enable the computation of simple commands in a different system or on a different machine, but it will also serve as a means of conveying constituents of larger, more complex, computations.

The protocol is XML-based; in particular, the protocol messages are in the OpenMath language, and its TCP-sockets based implementation uses XML processing instructions to delimit these messages and convey small pieces of information on a higher level. Communication takes place using port 26133, reserved for SCSCP by the Internet Assigned Numbers Authority (IANA). At the moment of writing the protocol has reached version 1.3 and both client and server implementations exist in GAP, KANT, Maple, and MuPAD. The TRIP system also supports the protocol, using their own publicly available implementation of SCSCP [3]. Moreover, we have developed a Java library `org.symcomp.scscp`

[14] to facilitate third party developers in exposing their own applications using SCSCP.

Apart from two OpenMath Content Dictionaries accompanying the SCSCP protocol [10,11] several other Content Dictionaries were developed in the project, concerning for example efficient matrix representations or polynomial factorization. We expect to submit these to the OpenMath community for consideration in the Summer of 2009.

2   POPCORN – A Tasty OpenMath Representation

When handling OpenMath objects, one frequently finds oneself typing and reading lots of `OMA`s, `OMS`s, and so on. This may lead one to the conclusion that humans were not designed to parse XML. Therefore, whenever people discuss their experiences with OpenMath, they tend to use more human-readable shortcuts, often inspired by LATEXor a Maple-like syntax.

That is why we decided to produce an OpenMath representation taking this into account, and created POPCORN, which is an acronym standing for "Possibly Only Practical Convenient OpenMath Replacement Notation". For the sake of typographic beauty, we write it as "Popcorn".

We emphasize that Popcorn is merely an OpenMath representation that we consider convenient for humans, similar to the XML representation that is obviously more convenient for machines. Furthermore, if a two-dimensional environment such as a web browser is available, more sophisticated editors such as the MathDox formula editor [5] are even better. However, we still think Popcorn is a valuable addition, e.g. for quick tests, command line applications, etc.

Parsing of Popcorn is sufficiently fast for small examples, but for larger OpenMath trees the XML representation can be parsed more efficiently, if only because when parsing Popcorn code an intermediate abstract syntax tree has to be constructed, while such a tree is inherent to the XML representation.

The Popcorn language itself not easily user-extensible, but because the Popcorn grammar is included in the libraries, an advanced user may change the grammar, e.g. add infix operators, special symbols, etc, and use it in his or her own application.

2.1   Elementary OpenMath in Popcorn

We first look at the notation used for the elementary OpenMath objects.

Integers are typed just as one expects: as decimal numbers without whitespace inside or prefixed with `0x` in hexadecimal representation;
Floats are typed either as, e. g. `2.34e12` or `0f###` where the `#` represent hexcharacters as in the `hex` attribute of `OMF`;
Strings are wrapped in `"` or `'` ;
References are given either in the simple form `#name` (for local references `<OMR href="#name"/>`) or the more complex form `##http://somewhere/something/##` (for non-local references);

Variables are whitespace-free strings prefixed with `$`;
Symbols are written as `cdname.name`.

To add an id value to any object, simply postpone it with `:theid`.

2.2   Compound OpenMath in Popcorn

Application is encoded by postponing the parenthesized arguments to the applied object, e.g. `arith1.plus(1,2,3)`;

Binding is done by typing square brackets behind the bound object. Within the brackets the comma-separated bound variables are separated from the expression by `->`, e.g. `quant1.forall[$x, $y -> ...]`;

Attribution is done by adding key/value-pairs as a comma-separated list in braces to the attributed object, e.g. `1.2{aa.bb -> "cc", "dd" -> 3}`

2.3   Syntactically Sugared/Salted Popcorn

To allow for more intuitive notation, we added some shortcuts:

For the `arith1` symbols `plus`, `times`, and the `relation1` symbols we added the obvious infix symbols `+`, `*`, `=`, `<`, `<=`, and so on. The same is true for the `logic1` symbols, all with a well-defined operator precedence.

For a reasonably large number of frequently used symbols, we decided to get the cdname-free name into the global context, e.g., `sum`, `sin`, `true`, `lambda`, `pi`, `i`, etc.

To construct a `list1.list`, one may simply use square brackets, and to construct `set1.set`, braces can be used (The Popcorn parser automatically checks whether for example an expression in square brackets matches the binding pattern, so that no confusion arises). Constructing `nums1.rational` can be done by separating numerator and denominator with `//`. Similarly `nums1.complex_cartesian` can be constructed by separating the real and imaginary part with `|`.

Also, some of the functionality of the experimental `prog1` Content Dictionary is exposed in a Maple-like syntax.

2.4   Popcorn Examples

```
sin(3)                    <OMA>
                            <OMS cd="arith1" name="sin">
                            <OMI>3</OMI>
                          </OMA>

lambda[$x->1+$x]          <OMBIND>
                            <OMS cd="fns1" name="lambda" />
                            <OMBVAR>
                              <OMV name="x" />
                            </OMBVAR>
                            <OMA>
                              <OMS cd="arith1" name="plus" />
                              <OMI>1</OMI>
                              <OMV name="x" />
                            </OMA>
                          </OMBIND>
```

```
$a := [1//2, (2|8):x]          <OMA>
                                 <OMS cd="prog1" name="assign" />
                                 <OMV name="a" />
                                 <OMA>
                                   <OMS cd="list1" name="list" />
                                   <OMA>
                                     <OMS cd="nums1" name="rational" />
                                     <OMI>1</OMI><OMI>2</OMI>
                                   </OMA>
                                   <OMA id="x">
                                     <OMS cd="nums1" name="complex_cartesian" />
                                     <OMI>2</OMI><OMI>8</OMI>
                                   </OMA>
                                 </OMA>
                               </OMA>
1.2{aa.bb -> "cc"}             <OMATTR>
                                 <OMATP>
                                   <OMS cd="aa" name="bb" />
                                   <OMSTR>cc</OMSTR>
                                 </OMATP>
                                 <OMF dec="1.2" />
                               </OMATTR>
```

Here another motivation for the name Popcorn can be seen – it turns something pretty small into something rather giant.

3   `org.symcomp.openmath` – Convenient Handling of OpenMath with Java

For the development of tools and applications within SCIEnce, Java seemed a natural choice because of its portability and the availability of many libraries. Although there are some Java OpenMath Libraries available [7,8], these are older (last update in 2000 and 2004, respectively) and we disagreed with some of the design choices made.

We therefore created a new library that takes advantage of the recent developments in Java, such as annotations and generics, and we designed it from the ground up to be as easily extensible as possible. It provides many convenience classes and handy methods to traverse, construct, and analyze OpenMath trees. Furthermore, it has completely transparent support for OpenMath Attributions, eliminating the need to handle these objects in any special way.

Import and export to OpenMath 2 XML, OpenMath 2 Binary, and Popcorn are included. Moreover, we have implemented export to LATEX to demonstrate the great extensibility of the library, and because it easily enables rendering of OpenMath in browsers using the well known `jsMath` package. We expect to include Strict Content MathML 3 soon as well, in view of the recent developments with respect to OpenMath 3.

We hope to clarify the simplicity and elegance this library offers by means of the small example in Listing 1.1 (the usual Java preliminaries have been omitted).

3.1   Custom Renderers

To feed OpenMath data into other applications, it is often necessary (or at least convenient) to produce a specific format. This is wired into

```
 1 //Creating an OpenMath application
   OMSymbol s = new OMSymbol("somecd", "add");
   assert s.isSymbol("somecd", "add");
   OpenMathBase[] params = new OpenMathBase[] {
       new OMInteger(1),
 6     new OMString("lala")
   };
   OMApply oma = s.apply(params);
   assert oma.isApplication("somecd", "add");

11 //Creating the same OpenMath object from Popcorn
   OpenMathBase oma2 = OpenMathBase.parse("somecd.add(1, 'lala')");
   assert oma.equals(oma2);

   //Creating an OpenMath object from the XML representation
16 OpenMathBase omi = OpenMathBase.parse("<OMOBJ><OMI>42</OMI></OMOBJ>");
   assert omi.deOMObject().isInteger(42);

   //Creating an OpenMath Binding, using a combination of pure Java
   //  creations and Popcorn.
21 OMVariable[] omvs = new OMVariable[] { new OMVariable("x") };
   OMBind ombind = s.bind(omvs, OpenMathBase.parse("$x + 1"));
   assert ombind.isBinding(s);

   //Convenient equality testing (note that it is a literal comparison,
26 //  e.g. alpha-conversion is not included)
   assert ombind.toPopcorn().equals("cdname.name[$x -> $x + 1]");
```

Listing 1.1. Using the `org.symcomp.openmath` library

`org.symcomp.openmath` as *custom renderers*. We designed these classes in such a way that producing e.g. a renderer for the Magma language took only a few lines of code.

The LATEX- and Popcorn-renderer are made using the same mechanism. These also give the user a great starting point for developing his/her own custom renderer.

4   Conclusion

In this short communication we have given an overview of the current developments in the European project "SCIEnce," in particular the implementation of the OpenMath based SCSCP protocol. We presented two Java libraries assisting this implementation, one for conveniently handling OpenMath objects, the other for executing the SCSCP protocol itself. These libraries enable a developer to expose his own application to other systems using OpenMath and SCSCP, requiring nothing but the strictly necessary from that developer.

Future activities include extending OpenMath support in the participating systems, porting the libraries to C++, both for developers using C or C++ as well as for improved performance, and adding MathML support to the library once OpenMath3 and MathML3 have been finalized.

5   License and Availability

The `org.symcomp.openmath` library and the SCSCP library `org.symcomp.scscp` are released under the Apache 2 License. In February 2009 the first public

release was made [14]. The libraries are available as binaries, source packages or they may be used as Maven [4] dependencies. Available on the website is also a comprehensive (and continuously improving) API documentation.

Part of this release is an extensive example that uses both the OpenMath and SCSCP library, and shows how little is needed to use the libraries to set up SCSCP clients and servers.

References

1. Caprotti, O., Cohen, A.: Connecting proof checkers and computer algebra using OpenMath. In: Bertot, Y., Dowek, G., Hirschowitz, A., Paulin, C., Théry, L. (eds.) TPHOLs 1999. LNCS, vol. 1690, p. 109. Springer, Heidelberg (1999)
2. Caprotti, O., Cohen, A.M., Riem, M.: Java Phrasebooks for Computer Algebra and Automated Deduction. SIGSAM Bulletin (2000) (Special Issue on OpenMath)
3. Gastineau, M.: SCSCP C Library - A C/C++ library for Symbolic Computation Software Composibility Protocol, IMCCE (2009), `http://www.imcce.fr/Equipes/ASD/trip/scscp/`
4. Maven: A software project management and comprehension tool, `http://maven.apache.org/`
5. MathDox OpenMath Formula Editor, `http://mathdox.org/formulaeditor/`
6. OpenMath, `http://www.openmath.org/`
7. PolyMath/OpenMath, `http://pdg.cecm.sfu.ca/openmath/`
8. RIACA OpenMath Library, `http://www.mathdox.org/new-web/openmath.html`
9. Freundt, S., Horn, P., Konovalov, A., Linton, S., Roozemond, D.: Symbolic Computation Software Composability Protocol (SCSCP) specification, Version 1.2 (2008), `http://www.symbolic-computation.org/scscp/`
10. Roozemond, D.: OpenMath Content Dictionary: scscp1, `http://www.win.tue.nl/SCIEnce/cds/scscp1.html`
11. Roozemond, D.: OpenMath Content Dictionary: scscp2, `http://www.win.tue.nl/SCIEnce/cds/scscp2.html`
12. Symbolic Computation Infrastructure for Europe, `http://www.symbolic-computation.org/`
13. Freundt, S., Horn, P., Konovalov, A., Linton, S., Roozemond, D.: Symbolic Computation Software Composability. In: Autexier, S., Campbell, J., Rubio, J., Sorge, V., Suzuki, M., Wiedijk, F. (eds.) AISC 2008, Calculemus 2008, and MKM 2008. LNCS, vol. 5144, pp. 285–295. Springer, Heidelberg (2008)
14. Homepage of the `org.symcomp.openmath` and `org.symcomp.scscp` libraries, `http://java.symcomp.org/`