OpenMath in SCIEnce: Evolving of Symbolic Computation Interaction

Sebastian Freundt[1], Peter Horn[2], Alexander Konovalov[3], Sylla Lesseni[1], Steve Linton[3], and Dan Roozemond[4]

[1] Fakultät II - Institut für Mathematik, Technische Universität Berlin, Berlin, Germany, {freundt|lesseni}@math.tu-berlin.de
[2] Fachbereich Mathematik, Universität Kassel, Kassel, Germany, horn@math.uni-kassel.de
[3] School of Computer Science, University of St Andrews, Scotland, {alexk|sal}@mcs.st-and.ac.uk
[4] Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, Netherlands, d.a.roozemond@tue.nl

Abstract. We present SCSCP – the Symbolic Computation Software Composability Protocol. SCSCP is a remote procedure call framework for computational algebra systems in which both data and protocol instructions are encoded in the OpenMath language. We present SCSCP implementations in several CASes and other SCSCP-compliant applications and APIs, developed with the support of the EU FP6 project "SCIEnce – Symbolic Computation Infrastructure for Europe".

1 Combining Symbolic Computation Systems

Many research problems which could be tackled with computer algebra systems (CASes) cannot be solved within a single system or could be solved much faster if a combination of two or more CASes would allow performing each step in the fastest available implementation.

Examples include number theory computations in a specialized system, symbolic calculations on generic character tables, computations on large finite state automata, Gröbner computations that are faster in one system than in another, libraries or plug-ins that are available in a system on Linux but not on Windows, etc.

In this paper we are giving an overview of the SCIEnce project activity to develop a framework which enables efficient and reliable combining of CASes for such purposes. Another area the SCIEnce project hopes to bring improvement to is that of libraries and databases of mathematical objects that may be stored in some universal format accessible to many systems.

Since the OpenMath standard emerged, a lot of work was done on combining CASes, e.g. the MONET project [8] and the various translators (phrasebooks) produced by RIACA [14] (see also the OpenMath webpage [10] for a detailed account of OpenMath software and tools). However, the approach has often been

to have a system provide OpenMath by creating wrapper software that communicates with the system in the background and that performs the translation from OpenMath to the internal syntax and vice versa.

The approach taken in the SCIEnce project, however, is to build the OpenMath support into the systems themselves instead of creating a wrapper, which yields a much more robust implementation. Also, unlike in another well-known project in this area, namely Sage [17], the approach in SCIEnce is not to subordinate the packages from an integrating system, but to define an interface that any system can implement to provide a way to use the capabilities of other systems within a familiar environment.

We therefore set up a project aiming to define standards and construct an extensible framework which would:

- ensure seamless communication between CASes, both local and remote;
- use a universal format not relying on the particular input/output format for each system;
- ensure that each system implementing the standard can immediately offer services to and consume services from other such systems.

2 SCSCP and its CDs

To simplify the communication between the various CASes, we have developed a protocol called the "Symbolic Computation Software Composability Protocol", abbreviated SCSCP [3, 2]. 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 key features of SCSCP are:

- Mathematical data are encoded in OpenMath;
- The protocol messages are encoded in OpenMath as well, so that participating systems need to support only one language;
- The OpenMath support is wired directly into the joining systems. This is much more robust, easier to create, and faster than the usual practice of producing wrapping programs to enable OpenMath support.

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).

SCSCP does not require statefulness on behalf of the server, although it does offer support for working with so-called "remote objects." Such object can be created on the server (thus, changing its state) and used in further computations as arguments of procedure calls.

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.

We will detail these implementations in Section 3, except for the Maple system which plans to announce its tools elsewhere at a later stage.

The protocol is also supported by TRIP, a general computer algebra system dedicated to celestial mechanics, using an own publicly available implementation of SCSCP [4]. Moreover, we have developed a Java library `org.symcomp.scscp` to facilitate third party developers in exposing their own applications using SCSCP. We provide details on this library in Section 4. Additionally, while some of our SCSCP implementations provide straightforward functionality for parallel computations, another result of the SCIEnce project is the SymGrid-Par middleware, which orchestrates computational algebra components into a parallel application that uses SCSCP for internal communication [19].

Apart from two OpenMath Content Dictionaries accompanying the SCSCP protocol [15, 16], several other Content Dictionaries were developed in the project, concerning, for example, polynomial factorization and efficient OpenMath representations of matrices, number fields and orders in number fields. We have submitted these content dictionaries separately to the OpenMath 2009 workshop.

As a simple example, we demonstrate a simple SCSCP session on the server. The server is running GAP and provides a procedure to identify a finite group in the GAP Small Groups Library. After the server receives an incoming connection, it replies with the connection initiation message. After that, the client replies with its preferred version, and the server confirms this version to the client.

```
S: <?scscp service_name="GAP" service_version="4.dev" service_id="
     localhost:26133:7617" scscp_versions="1.0 1.1 1.2 1.3" ?>
C: <?scscp version="1.3" ?>
S: <?scscp version="1.3" ?>
```

Then the client sends the procedure call to identify the cyclic group of order two given as permutation group:

```
C: <?scscp start ?>
   <OMOBJ>
       <OMATTR>
           <OMATP>
               <OMS cd="scscp1" name="call_id"/>
               <OMSTR>scscp.symcomp.org:26133:7617:eBFyqFae</OMSTR>
               <OMS cd="scscp1" name="option_return_object"/>
               <OMSTR></OMSTR>
           </OMATP>
           <OMA><OMS cd="scscp1" name="procedure_call"/>
               <OMA><OMS cd="scscp_transient_1" name="WS_IdGroup"/>
                   <OMA><OMS cd="permgp1" name="group"/>
                       <OMS cd="permutation1" name="right_compose"/>
                       <OMA><OMS cd="permut1" name="permutation"/>
                           <OMI>2</OMI>
                           <OMI>1</OMI>
                       </OMA>
                   </OMA>
               </OMA>
           </OMA>
       </OMATTR>
   </OMOBJ>
   <?scscp end ?>
```

The server responds that the group has catalogue number $[2, 1]$:

```
S: <?scscp start ?>
   <OMOBJ>
       <OMATTR>
           <OMATP>
               <OMS cd="scscp1" name="call_id"/>
               <OMSTR>scscp.symcomp.org:26133:7617:eBFyqFae</OMSTR>
           </OMATP>
           <OMA><OMS cd="scscp1" name="procedure_completed"/>
               <OMA><OMS cd="list1" name="list"/>
                   <OMI>2</OMI>
                   <OMI>1</OMI>
               </OMA>
           </OMA>
       </OMATTR>
   </OMOBJ>
   <?scscp end ?>
```

After that the client closes the connection, and the server is ready to accept new procedure calls.

3 Computer Algebra Systems

In this section we give a brief overview of the status of the SCSCP implementation in various systems.

3.1 GAP

In the GAP system, the support of OpenMath and SCSCP is implemented in two GAP packages with the same names.

The OpenMath package [1] provides an OpenMath phrasebook for GAP: it is responsible for the conversion from OpenMath to GAP and vice versa and reading/writing OpenMath objects from/to streams. The package provides a framework, allowing users to extend it with private content dictionaries.

The SCSCP package [7] implements the Symbolic Computation Software Composability Protocol on top of the GAP packages OpenMath, IO and GAP-Doc. The package has two main components: server and client. The server may be started interactively from the GAP session or as a GAP daemon. When the server accepts a connection from the client, it starts the "accept-evaluate-return" loop:

- accepts the `"procedure_call"` message;
- performs lookup of the appropriate GAP function;
- evaluates the result (or produces a side-effect);
- returns the result in the `"procedure_completed"` message or returns an error in the `"procedure_terminated"` message.

The SCSCP client performs the following basic actions:

- establishes connection with the specified server at the specified port;
- sends the `"procedure_call"` message to the server;

– waits for the result of the computation or returns to pick it up later;
– fetches the response, extracting the result from the `"procedure_completed"` message or entering the break loop in the case of the `"procedure_terminated"` message.

On top of this functionality we built a set of instructions for parallel computations using the SCSCP framework, allowing to send several procedure calls in parallel and then collect all results or pick up the first available result, and implemented the master-worker parallel skeleton.

To give the users an opportunity to test the package we are running a demo SCSCP server accessible at `chrystal.mcs.st-andrews.ac.uk`, port 26133. It is working under the development version of the GAP system and a selection of currently redistributed GAP packages. See the package homepage [7] for further information, downloads and documentation with examples.

3.2 KANT

The KANT system provides two main packages for SCSCP support:

– `libkant` package which contains the core functionality of the KANT system;
– `autokash` which consists of the KANT SCSCP server, a simple client and a server. The server is started by running the KANT/KASH daemon, named `kashd`, which is the main binary in the `autokash` package.

Here are the different steps when the connection from a client is accepted by the server:

– a socket is open and the SCSCP message comes in;
– the `"procedure_call"` part of the message is extracted;
– a table of xpaths is matched against the message and a callback function is looked up;
– after evaluation, the result (or an error message in case of error) is sent back to the client.

We are running two KANT SCSCP servers accessible at port 26133 at addresses `issel.math.tu-berlin.de` and `stirling.math.tu-berlin.de` and running under the recent development version of the KANT system. The users can also download the `autokash` package which contains `libkant` library from [6].

The OpenMath support is implemented in the `autokash` package. It is contained in the `openmath.la` library. To use that library, one should load it when running the KANT/KASH daemon `kashd`. See the public homepage [6] for more information.

The KANT SCSCP Client Shell: kapy We are developing at the present moment a KANT SCSCP client shell, named kapy. It is written in python and fully supports the SCSCP protocol. Also, the idea of using kapy is to ease manual typing when handling OpenMath objects. To connect from kapy to a running KANT SCSCP server, we need at least python version 2.5 and the script

`kapy.py`. The call `cas = kapy.connect(`*host*`, `*port*`)` will establish the connection with the SCSCP server running at the appropriate host and port. From then, one can create the openmath objects like OMI, OMF, OMSTR, OMV using:

- `cas.compute(kapy.omi(`*int*`))` for the intergers;
- `cas.compute(kapy.omf(`*float*`))` for the floats;
- `cas.compute(kapy.omf(`*str*`))` for the strings;
- `cas.compute(kapy.omv(`*var*`))` for the unused variables since the KANT system can not handle the symbolic objects.

The openmath objects OMS and OMA are constructed as follows:

- `cas.compute(kapy.oms(`*cdname*`, `*symbolname*`))` for OMS;
- `cas.compute(kapy.oma(`*cdname*`, `*symbolname*`, `*args*`))` for OMA.

We can also store the result in a variable to reuse it later or in other computations. Finally, since kapy is written in python, it is natural to be able to convert basic OM objects like OMI, OMF, OMSTR to the python representation using a function named `PyConvert`.

3.3 MuPAD

There are two main aspects for MuPAD SCSCP support:

- `OpenMath` MuPAD package;
- SCSCP server wrapper for MuPAD.

While the former offers the ability to parse, generate, and handle OpenMath in MuPAD, and to consume SCSCP services, the latter enables access to MuPADs mathematical abilities as an SCSCP service. Sadly, however, the current MuPAD end-user license agreement does not generally allow this. Therefore, below we concentrate on the `OpenMath` package.

To use the package, download it from [9] and put it into your `PACKAGEPATH`. It can then be loaded using `package("OpenMath")`. Afterwards, documentation is available through `OpenMath::doc()`.

The OpenMath Elements To represent the different OpenMath tags, there are the following constructors:

- `OpenMath::Apply(`*head*`, [`*params*`])` – expands to a function call of *head* on *params*;
- `OpenMath::Bind(`*head*`, [`*vars*`], `*expr*`)` – expands to a function call of *head* on *vars* and *expr*;
- `OpenMath::Error(`*head*`, [`*params*`])` – expands to an error textually containig *head* and *params*;
- `OpenMath::Float(`x`)` – expands to a `DOM_FLOAT`;

– `OpenMath::Integer(`*i*`)` – expands to a `DOM_INT`;
– `OpenMath::Object(`*o*`)` – expands to *o*;
– `OpenMath::Reference(`*id*`)` – expands to either the element with the given *id* or an `error`;
– `OpenMath::String(`*str*`)` – expands to the given `DOM_STR`;
– `OpenMath::Symbol(`*cdname*`, `*name*`)` – expands to either some MuPAD object or the MuPAD identifier ‘*cdname*`.`*name*‘;
– `OpenMath::Variable(`*name*`)` – expands to an unused `DOM_IDENT`.

All constructors accept an optional last argument *id* to set the id. A tree of these objects may be translated to MuPAD objects by calling `expand` on it. It may be necessary to call `eval` to get an actual result.

Calling `OpenMath::toXml` on an OpenMath tree gives a tree of `adt::XML` nodes representing it. This can then printed or converted to an XML string. All these constructors have a `doc` slot, so you can get further information by calling, e. g., `OpenMath::Apply::doc()`.

The OpenMath Parser In this domain, there are two functions available to turn an OpenMath XML string into a tree of `OpenMath::` objects as above:

– `OpenMath::parse(`*str*`)` – parses the string *str*;
– `OpenMath::parseFile(`*fname*`)` – reads and parses the file named *fname*.

Generating OpenMath With `generate::OpenMath`, a MuPAD expression can be converted into its OpenMath representation. Internally, the above mentioned `OpenMath` elements are used to assemble the result.

The result of the call to `generate::OpenMath` is always wrapped in an `OpenMath::Object`. To obtain the OpenMath representation without the wrapping `OMObject`, one can simply use `OpenMath(...)`.

SCSCP Client Connection By the call `s := SCSCP(`*host*`, `*port*`)` an SCSCP connection object is created, which can then be used to send commands to the SCSCP server. Note that the actual connection is initiated on construction by starting the java program WUPSI (see 4.4) which is bundled into the `OpenMath` package. It is using an asynchronous file system based message exchange mode and thus can be used to do computations in the background.

To actually let the server compute something, one uses `s::compute(...)` or, equivalently `s(...)`. Note that it may be necessary to wrap the parameter in `hold(...)` to prevent premature evaluation in MuPAD.

To use the connection asynchronously, the commands `send` and `retrieve` are used: `a := s::send(...)` returns an integer which may be used to identify the computation and to retrieve the result later with `s::retrieve(a)`. `retrieve` returns `FAIL` if the result of the computation is not yet computed unless you specify a second parameter `TRUE`. In that case the call will block until the result is ready.

To disconnect the client after use (and before e. g. `reset`) one can use the command `s::close()` to stop the corresponding Java program, and thus clean up the associated resources. Obviously, when MuPAD exits, this is done automatically.

4 Java SCSCP API

This Java library is intended to enable third party developers to use SCSCP servers (i.e. have their own application acting as an SCSCP client), or easily expose their own applications as SCSCP servers (i.e. have their own application acting as an SCSCP server).

The library has two essential parts: The OpenMath library `org.symcomp.openmath` and the SCSCP implementation `org.symcomp.scscp`. We will detail these libraries in Sections 4.2 and 4.3, respectively. Furthermore, the library comes with several examples that should serve as a good starting point for the user.

We use these libraries ourselves as well: In the MuPAD client and server application (Section 3.3), in an experimental MAGMA server, in the Webproxy (Section 4.5) and in WUPSI (Section 4.4).

4.1 The Popcorn 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. That is why we decided to create an OpenMath representation taking this into account, and created POPCORN. It 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 binary 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 are even better. However, we still think Popcorn is a valuable addition, e.g. for quick tests, command line applications, etc.

Popcorn is described in more details in the MKM 2009 paper [5].

4.2 `org.symcomp.openmath`

Although there are some Java OpenMath Libraries available [12, 13], 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, and export to LaTeX are included. Moreover, to feed OpenMath data into other applications, it is often necessary to produce a specific format. This is wired into `org.symcomp.openmath` as *custom renderers*. We designed this part of the library in such a way that producing e. g. a renderer for the MAGMA language took only a few lines of code. Moreover, 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.

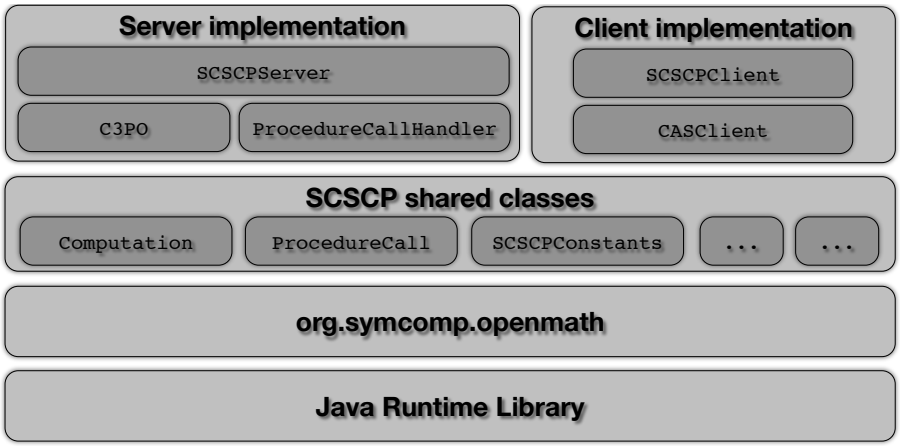

Fig. 1. The structure of the `org.symcomp` Java libraries

4.3 `org.symcomp.scscp`

As mentioned above, this library was designed to enable a third party developer to easily expose his or her own application using SCSCP. This library obviously uses `org.symcomp.openmath` for handling the OpenMath objects that are inherent to the SCSCP protocol.

The structure of the libraries is shown in Figure 1. Depending on the level of control a developer wants to have on the inner workings of the SCSCP protocol, he would extend a particular class.

For an application to use an SCSCP service such as GAP or MuPAD, a developer would typically subclass the `SCSCPClient`. She or he would then need to do little else than specify the host and port to connect to, phrase the relevant questions in OpenMath, and call the `compute` method.

If on the other hand someone wants to make his or her own application available for other SCSCP clients, she or he would typically subclass called the

`ProcedureCallHandler`. In its `handle` method, little else is required than interpreting the incoming OpenMath message, converting it to the internal format of the parent application, computing the solution, and converting the solution back to OpenMath.

4.4 WUPSI : a proof-of-concept example

The API described in this section allowed us to easily create WUPSI ("Wonderful Universal Popcorn SCSCP Interface"). It is a small command-line application that allows to connect to one or more SCSCP servers and issue computation requests.

It was designed with two main purposes in mind. Firstly, it serves as a great debugging tool for SCSCP implementations, as one simply enters OpenMath, and receives OpenMath back (possibly in the form of Popcorn). Secondly, it is an extensive example of how the Java libraries may be used, and could serve as a nice reference for those who want to use these libraries.

Apart from the example uses shown in Listing 2 many other more advanced functions are available, such as a poor-man's parallelization tool and the possibility to have WUPSI act as an SCSCP proxy server for connected systems.

4.5 WebProxy

The WebProxy is a Java application meant as an administration and orchestration console for one or more SCSCP compliant services. Whilst the WebProxy is a web application intended for direct user interaction, it also provides access to the capabilities of the connected CASes through SOAP and GET/POST interfaces.

5 Licensing and Availability

The GAP packages are distributed under the GNU Public License as well as the GAP system itself and are available from their sites and from the GAP homepage `http://www.gap-system.org`.

The KANT system and the KANT SCSCP packages are distributed under the GNU Public Licence and are available from the KANT website `http://www.math.tu-berlin.de/~kant/`.

The MuPAD package is not provided by the makers of MuPAD, SciFace Software, but by the University of Kassel. It is published [9] under an Apache 2 License and should be compatible with MuPAD 4 and above.

The SCSCP library `org.symcomp.scscp` and the `org.symcomp.openmath` library are released under the Apache 2 License. In February 2009 the first public release was made [11]. The libraries are available as binaries, source packages or they may be used as Maven dependencies. Available on the website is also a comprehensive (and continuously improving) API documentation. WUPSI will be available for download from [11] shortly.

```
WUPSI 1.2 -- Wonderful Universal Popcorn SCSCP Interface
(c) 2009 D. Roozemond & P. Horn

WUPSI[n/a]0> connect some.server:26139 as gap
# connected to 'some.server' on port '26139' using symbolic name 'gap'
# Service Info: service Name 'GAP', service version '4.dev'

WUPSI[gap]0> 126+2323*232
539062

WUPSI[gap]1> local $a := $_out0
# Stored this in local variable '$a':
539062

WUPSI[gap]2> connect 127.0.0.1:26134 as mupad
# connected to '127.0.0.1' on port '26134' using symbolic name 'mupad'
# Service Info: service Name 'MuPAD', service version '0.6.0-mupad
    -5.2.0'

WUPSI[mupad]2> output format latex
# switched output format to LATEX.

WUPSI[mupad]2> sum(1 .. infinity, lambda[$x -> 1/$x^2])
{\pi}^{2} \cdot \frac{1}{6}

WUPSI[mupad]3> output format popcorn
# switched output format to POPCORN.

WUPSI[mupad]3> local $p := 2^127-1
# Stored this in local variable '$p':
170141183460469231731687303715884105727

WUPSI[magma]4> use gap
# switched to system with symbolic name 'gap', service Name 'GAP',
    service version '4.dev'.

WUPSI[gap]4> $p-2^101*$a
168774498924748772136428072069291311103

WUPSI[gap]4> describe arith1.plus
# -- Description for 'arith1.plus' --
The symbol representing an n-ary commutative function plus.
# -- END description for 'arith1.plus' --
```

Listing 2. Using WUPSI

For the links to the most recent available downloads see the homepage of the SCIEnce project `http://www.symbolic-computation.org/`

6 Conclusions and future work

In this paper we presented SCSCP - a simple light-weight OpenMath-based remote procedure call framework, and gave an overview of SCSCP-compliant applications, represented by computer algebra systems, middleware and APIs. Further information and concrete examples may be found in (mostly available online) documentation for appropriate tools.

Among our future directions are developments of new content dictionaries additionally to those submitted to the OpenMath 2009 workshop, and increasing

the support of binary OpenMath format which seems inevitably needed in really large-scale computations.

We hope that the appearance of SCSCP and examples of its use in our CASes stimulate developers of other systems to support the OpenMath format to exchange mathematical data, and we hope more SCSCP-compliant software will become available in the future. We would like to strengthen this invitation by offering our support and advice to all interested parties.

References

1. M. Costantini, A. Konovalov and A. Solomon. *GAP package OpenMath.* Version 10.0, 2009. `http://www.cs.st-andrews.ac.uk/~alexk/openmath.htm`
2. S. Freundt, P. Horn, A. Konovalov, S. Linton and D. Roozemond. Symbolic Computation Software Composability. In *Intelligent Computer Mathematics, AISC/Calculemus/MKM 2008 proceedings*, Lecture Notes in Computer Science 5144/2008, Springer, p.285-295.
3. S. Freundt, P. Horn, A. Konovalov, S. Linton, D. Roozemond, Symbolic Computation Software Composability Protocol (SCSCP) specification, Version 1.3, 2009. `http://www.symbolic-computation.org/scscp/`
4. M. Gastineau, *SCSCP C Library - A C/C++ library for Symbolic Computation Software Composibility Protocol*, IMCCE, 2009, `http://www.imcce.fr/Equipes/ASD/trip/scscp/`
5. Peter Horn and Dan Roozemond. OpenMath in SCIEnce: SCSCP and POPCORN. To appear in *Intelligent Computer Mathematics, MKM 2009 proceedings.*
6. *KANT SCSCP Package.* `http://www.math.tu-berlin.de/~kant/kantscscp.html`
7. A. Konovalov and S. Linton. *GAP package SCSCP.* Version 1.1, 2009. `http://www.cs.st-andrews.ac.uk/~alexk/scscp.htm`.
8. *The MONET project.* `http://monet.nag.co.uk/monet/`
9. *MuPAD OpenMath Package.* `http://mupad.symcomp.org/`
10. *OpenMath.* `http://www.openmath.org/`
11. The `org.symcomp.openmath` and `org.symcomp.scscp` libraries: `http://java.symcomp.org/`
12. *PolyMath/OpenMath.* `http://pdg.cecm.sfu.ca/openmath/`
13. *RIACA OpenMath Library.* `http://www.mathdox.org/new-web/openmath.html`
14. *RIACA OpenMath Phrasebooks.* `http://www.mathdox.org/new-web/products.html`
15. Roozemond, D.: *OpenMath Content Dictionary: scscp1.* `http://www.win.tue.nl/SCIEnce/cds/scscp1.html`
16. Roozemond, D.: *OpenMath Content Dictionary: scscp2.* `http://www.win.tue.nl/SCIEnce/cds/scscp2.html`
17. *Sage.* `http://sagemath.org/`
18. *Symbolic Computation Infrastructure for Europe.* `http://www.symbolic-computation.org/`
19. A.D. Al Zain, P.W. Trinder, K. Hammond, A. Konovalov, S. Linton and J. Berthold. Parallelism without Pain: Orchestrating Computational Algebra Components into a High-Performance Parallel System. In *International Symposium on Parallel and Distributed Processing with Applications*, 2008, p.99–112.