

# SYMBOLIC COMPUTATION SOFTWARE COMPOSABILITY PROTOCOL (SCSCP) SPECIFICATION

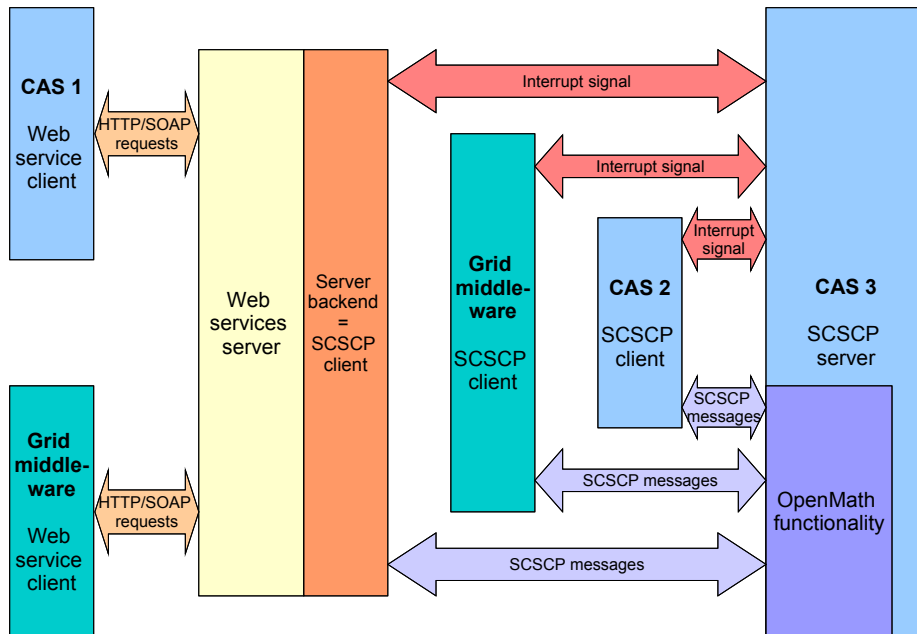
S. LINTON, A. KONOVALOV

## 1. INTRODUCTION

This document specifies the requirements for the software to be developed by the NA3 activity of the SCIENCE project for the subsequent usage in the NA3 and JRA1 activities.

Specifically it describes a protocol by which a CAS may offer services and a client may employ them. We envisage clients for this protocol including:

- A Web server which passes on the same services as Web services using SOAP/HTTP to a variety of possible clients;
- Grid middleware;
- Another instance of the same CAS (in a parallel computing context);
- Another CAS running on the same system.



This Specification is now replaced by the newer version, available as CIRCA preprint 2007/13. The project 026133 "SCIENCE - Symbolic Computation Infrastructure in Europe" is supported by the EU FP6 Programme.

Figure 1. SCSCP usage.

In section 2 we describe the meaning of all possible messages that can appear during communication between various software and the allowed sequences of such messages. In the section 3 we specify how these messages are encoded as OpenMath (or in one case WSDL objects). Finally in section 4 we describe one solution (suitable for UNIX systems, at least) for the practical problems of establishing a connection and delivering these messages. In the Appendix A we specify the list of necessary OpenMath symbols. Examples of OpenMath messages are given in the Appendix B, and an example of the Web Services Description Language message - in the Appendix C.

List of abbreviations used in the document:

**CD** - Content Dictionary

**OM** - OpenMath

**SCSCP** - Symbolic Computation Software Composability Protocol

**WS** - Web-service

**WSDL** - Web Services Description Language

## 2. SEMANTIC DESCRIPTIONS

### 2.1 Messages to CAS

#### 2.1.1. Procedure call

OM message containing the following information:

- *Procedure name* - the name of the procedure registered as a web-service;
- *Arguments* - arguments that will be passed to the procedure being called; (Remark: we treat procedure options, i.e. guidance options for used algorithms, as arguments as well);
- *Options/Attributes* - attributes and options that will specify the behaviour of the system:
  - call identifier;
  - type of action performed with the result:
    - \* storing the result at the CAS side and returning a cookie referring to it;
    - \* returning the result of the procedure (that may involve actual computation or retrieving previously stored result);
  - procedure runtime limit;
  - minimal/maximal memory limits;
  - debugging level, determining degree of output detail;
  - other options that might appear during the development and be system dependent.

Some examples of standard procedures:

- **STORE** - compute an object on the CAS side and store it there, returning only a cookie, i.e. a pointer to that object that is usable in the future to get access to the actual object;
- **RETRIEVE** - using a cookie, previously obtained using the **STORE** procedure, return to the client an OM object representing an object, referred by the cookie.

### 2.1.2. Interrupt signal

This signal be sent to the CAS at any time, and should be processed “out-of-band”. That is, it should be processed as soon as possible, and not wait behind other messages that may have been sent or received before it and are unprocessed.

The impact of this message is that the results of the computation currently being performed are not needed and so the CAS need not complete the computation. If the CAS chooses not to complete the calculation it should send a Procedure terminated message reporting this.

There are some obvious and undesirable race conditions associated with this message, since the computation being performed when it is received may not be the one the client expects to be being performed. Ideally the solution would be for this message to carry the identity of the procedure call to be interrupted, but it is significantly easier to deliver an out-of-band message with no content than one with content, so, for the time being, we ignore this problem. The client can, at least, detect when this has happened by watching the message stream from the CAS.

Note that it is always correct for a CAS to ignore an interrupt, and that this may be appropriate when procedure calls are quick. It is assumed that this option will be regulated by the WS provider.

## 2.2 Messages from CAS

### 2.2.1. Procedure completed

OM message containing the information about the result of the procedure:

- *Result value* - if the procedure returns result, it must be contained in this section of the message. If the procedure only produces side-effect, such section is not necessary, since this message itself acts as a signal about its successful completion;
- *Additional information*
  - call identifier;
  - procedure runtime;
  - memory used;
  - other information that we might need (may be system dependent).

### 2.2.2. Procedure terminated.

OM message that acts as a signal about procedure termination. It must contain the following debugging information:

- if termination was caused by the CAS, then it must carry the original message from the CAS running as a WS;
- if termination was caused by another standard error, then it must carry the description of this error, in particular:
  - can not compute OM object;
  - wrong cookie (the object does not exist);
  - terminated by interrupt;
  - procedure not supported;
  - ran out of resource (time or memory);
- call identifier;
- procedure runtime before termination;
- memory used (if available);
- other information that we might need (may be system-dependent).

### 2.2.3. List of procedures supported by WS.

WSDL message containing the list of supported procedures and describing how to access them. This can be used by a software system e.g. resource broker to determine whether the WS match the requirements or converted to human-readable description for the user. It must specify general characteristics of WS, such as its address, communication details etc., and also the information about each available procedure:

- procedure name;
- number of arguments;
- documentation elements, containing, among other useful information, such requirements as format of each argument (for example, an OM object as a string or as tree, CAS command as an OM string, a cookie for substitution, etc.).

### 2.3 Allowed sequences of messages

Once a connection has been established and any initial technical information exchanged, the mechanism for which is part of an implementation of this protocol and addressed in section 4, the CAS begins the conversation with a List of Supported Procedures message and may include additional such messages at any point.

Apart from this, and the interrupt message, of which more later, the communication from client to CAS is a stream of procedure calls and the response is a stream of procedure completed and/or procedure terminated messages. The client is permitted to send as many procedure calls as it like, subject to the buffering capabilities of the channel used in a particular implementation. The CAS must process them in sequence and send *either* a procedure completed, *or* a procedure terminated message (but not both) for each. As a convenience and to assist debugging, the calls and responses are also associated via the `call_ID` attribute.

The interrupt message can be sent to the CAS at any time and should be delivered and acted on immediately. It entitles the CAS to stop processing the current procedure call and respond to it with a suitable procedure terminated message.

## 3. TECHNICAL DESCRIPTIONS

### 3.1 Messages to CAS

#### 3.1.1. Procedure call

The procedure call is an OM object having in general case the following structure:

```
<OMOBJ>
  <OMATTR>
    <!-- beginning of attribution pairs -->
    <OMATP>
      <!-- call identifier (or it will be assigned by WS???) -->
      <OMS cd="cascall1" name="call_ID" />
      <OMSTR>call_identifier</OMSTR>
      <OMS cd="cascall1" name="option_runtime" />
      <OMI>runtime_limit_in_milliseconds</OMI>
      <OMS cd="cascall1" name="option_min_memory" />
      <OMI>minimal_memory_required_in_bytes</OMI>
      <OMS cd="cascall1" name="option_max_memory" />
      <OMI>memory_limit_in_bytes</OMI>
```

```

    <OMS cd="cascall1" name="option_debuglevel" />
    <OMI>debuglevel_value</OMI>
    <OMS cd="cascall1" name="option_return_object" />
    <!-- another possibility is "option_return_cookie" -->
    <OMSTR/>
</OMATP>
<!-- Attribution pairs finished, now the procedure call -->
<OMA>
    <OMS cd="cascall1" name="procedure_call" />
    <OMSTR>NameOfTheProcedureRegisteredAsWebService</OMSTR>
    <!-- Here we use OMOBJ tags as delimiters for arguments,
         but actually nested OMOBJ tags are not allowed.
         See examples in the Appendix B. -->
    <OMOBJ>Argument1</OMOBJ>
    <!-- ... -->
    <OMOBJ>ArgumentM</OMOBJ>
</OMA>
</OMATTR>
</OMOBJ>

```

Remarks:

1. The definition of the OM symbol `procedure_call` should state that its first OM argument the first OM object is always the name of the procedure registered as WS, and the remaining OM objects are its arguments in the required order.
2. OM symbols for options will be introduced accordingly to the list of options given in 2.1.1. If the necessity of new options will be discovered, new OM symbols for them must be added to the CD.
3. Options may be omitted in the procedure call, and in this case their default values must be used.

### 3.1.2. Interrupt signal

By design, this message carries no content, the CAS simply needs to be aware that it has received an interrupt.

## 3.2 Messages from CAS

### 3.2.1. Procedure completed

The procedure completion message is an OM object having in the most general case the following structure:

```

<OMOBJ>
  <OMATTR>
    <!-- beginning of attribution pairs, amount of which depends
         on debugging level, and may include procedure name,
         OM object for the original procedure call, etc. -->
  <OMATP>
    <OMS cd="cascall1" name="call_ID" />
    <OMSTR>call_identifier</OMSTR>
    <OMS cd="cascall1" name="info_runtime" />
    <OMI>runtime_in_milliseconds</OMI>
    <OMS cd="cascall1" name="info_memory" />

```

```

    <OMI>used_memory_in_bytes</OMI>
  </OMATP>
  <!-- Attribution pairs finished, now the result -->
  <OMA>
    <OMS cd="cascall1" name="procedure_completed" />
    <!-- The result itself, may be OM symbol for cookie -->
    <!-- Here we use OMOBJ tags as delimiters for the object,
          but actually nested OMOBJ tags are not allowed.
          See examples in the Appendix B. -->
    <OMOBJ>OM_object_corresponding_to_the_result</OMOBJ>
  </OMA>
</OMATTR>
</OMOBJ>

```

In case when the procedure returns the cookie, the returned OM object must have the following structure:

```

<OMOBJ>
  <OMATTR>
    <OMATP>
      <OMS cd="cascall1" name="call_ID" />
      <OMSTR>call_identifier</OMSTR>
    </OMATP>
    <OMA>
      <OMS cd="cascall1" name="procedure_completed">
      <OMR xref="CAS_variable_identifier" />
    </OMA>
  </OMATTR>
</OMOBJ>

```

### 3.2.2. Procedure terminated.

The procedure termination message is an OM object having in the most general case the following structure:

```

<OMOBJ>
  <OMATTR>
    <!-- beginning of attribution pairs, amount of which depends
          on debugging level, and may include procedure name,
          OM object for the original procedure call, etc. -->
    <OMATP>
      <OMS cd="cascall1" name="call_ID" />
      <OMSTR>call_identifier</OMSTR>
      <OMS cd="cascall1" name="info_runtime" />
      <OMI>runtime_in_milliseconds</OMI>
      <OMS cd="cascall1" name="info_memory" />
      <OMI>used_memory_in_bytes</OMI>
    </OMATP>
    <!-- end of attribution pairs -->
    <!-- now the application part of the OM object -->
    <OMA>

```

```

<OMS cd="cascall1" name="procedure_terminated" />
<OME>
  <OMS cd="cascall1" name="name_of_standard_error"/>
  <!-- Error description depends on error type -->
  <OMSTR>Error_message</OMSTR>
</OME>
</OMA>
</OMATTR>
</OMOBJ>

```

### 3.2.3. List of supported procedures.

The list of supported procedures is the WSDL message describing Web services and specifying how to access them. Its format must satisfy the WSDL 2.0 standard (<http://www.w3.org/2002/ws/desc/>). The WSDL message binds together information about all procedures provided by the WS.

Such details like WS address and communication details are specified via standard WSDL constructions intended for this purpose, and currently we omit them.

Procedures are described in `<types>` elements, where procedure names and their arguments are specified.

Other useful information in a human-readable form, including such requirements as format of each argument (for example, an OM object as a string or as tree, CAS command as an OM string, a cookie for substitution, etc.) is contained in `<documentation>` elements.

The message exchange pattern is specified by the `<interface>` element.

An example of the WSDL message is given in the Appendix C.

## 4. REFERENCE IMPLEMENTATION

In this section we describe a simple implementation suitable for use for local connections on UNIX systems. It may also be usable under Windows provided POSIX signals is available.

**4.1. Connection Initiation.** The CAS wishing to provide an SCSCP service should listen on port 26133 [our EU project reference number, seems as good a choice as any]. When a client connects, the CAS should send it a Connection Information message, and then commence an exchange of messages as described in section 2.3.

**4.1.1. Connection information Message.** This non-OpenMath formatted message conveys which version of SCSCP the CAS is expecting to use and the PID of the CAS process to which the interrupt signals should be directed. A value of 0 of the CAS PID indicates that no signals should be sent.

This message is formatted as the string `SCSCP_VERSION N CAS_PID xxxx`, followed by a zero byte, where: `SCSCP_VERSION` and `CAS_PID` are fixed strings; `N` and `xxxx` are decimal numbers.

**4.2. Ongoing message exchange.** All messages except interrupts are delivered as OpenMath objects in either the XML or the binary OpenMath encoding, transmitted via the socket connection.

**4.3. Interrupt.** The interrupt signal is delivered by sending SIGUSR2 to the CAS.

## 5. APPENDIX A.

**The list of OM symbols to be developed for the cascall1 CD**

- (1) Procedure call
- (2) Call identifier
- (3) Option names (separate OM symbol for each standard option: action performed with the result, runtime limit, memory limits, debugging level, etc.)
- (4) Procedure completed
- (5) Information messages (separate OM symbol for each standard portion of information: runtime, memory usage etc.; may be unified with those symbols for options)
- (6) Procedure terminated
- (7) Standard errors (separate OM symbol for each standard error: can not compute OM object, cookie refers to a non-existing object, terminated by interrupt, procedure not supported, out of resources etc.)

6. APPENDIX B.

Examples of OM messages

B.1 An example of the `procedure_call` message

`GroupIdentificationService` accepts permutation group  $G$  given by its generators and returns the `procedure_completed` message with the number of this group in the GAP Small Groups Library, or the `procedure_terminated` message groups of order  $|G|$  are not contained in that library or identification for groups of such order is not available in GAP.

```

<OMOBJ>
  <OMATTR>
    <OMATP>
      <OMS cd="cascall1" name="call_ID" />
      <OMSTR>alexk_9053</OMSTR>
      <OMS cd="cascall1" name="option_runtime" />
      <OMI>300000</OMI>
      <OMS cd="cascall1" name="option_min_memory" />
      <OMI>40964</OMI>
      <OMS cd="cascall1" name="option_max_memory" />
      <OMI>134217728</OMI>
      <OMS cd="cascall1" name="option_debuglevel" />
      <OMI>2</OMI>
      <OMS cd="cascall1" name="option_return_object" />
      <OMSTR/>
    </OMATP>
    <OMA>
      <OMS cd="cascall1" name="procedure_call" />
      <OMSTR>GroupIdentificationService</OMSTR>
      <!-- Argument 1 -->
      <OMA>
        <OMS cd="group1" name="group"/>
        <OMA>
          <OMS cd="permut1" name="permutation"/>
          <OMI> 2</OMI>
          <OMI> 3</OMI>
          <OMI> 1</OMI>
        </OMA>
        <OMA>
          <OMS cd="permut1" name="permutation"/>
          <OMI> 1</OMI>
          <OMI> 2</OMI>
          <OMI> 4</OMI>
          <OMI> 3</OMI>
        </OMA>
      </OMA>
      <!-- End of argument 1 -->
    </OMA>
  </OMATTR>
</OMOBJ>

```

In the next example we retrieve the group [24,12] from GAP Small Groups Library, creating it at the GAP side and requesting a cookie for it (omitted options will be set to default values):

```

<OMOBJ>
  <OMATTR>
    <OMATP>
      <OMS cd="cascall1" name="call_ID" />
      <OMSTR>alexk_9053</OMSTR>
      <OMS cd="cascall1" name="option_return_cookie" />
      <OMSTR/>
    </OMATP>
    <OMA>
      <OMS cd="cascall1" name="procedure_call" />
      <OMSTR>GroupByCatalogueNumber</OMSTR>
      <OMI> 24</OMI> <!-- Argument 1 -->
      <OMI> 12</OMI> <!-- Argument 2 -->
    </OMA>
  </OMATTR>
</OMOBJ>

```

In the next example we are sending an OM object containing MathML object:

```

<OMOBJ>
  <OMATTR>
    <OMATP>
      <OMS cd="cascall1" name="call_ID"/>
      <OMSTR>a1d0c6e83f027327d8461063f4ac58a6</OMSTR>
      <OMS cd="cascall1" name="option_return_cookie"/>
      <OMSTR/>
    </OMATP>
    <OMA>
      <OMS cd="cascall1" name="procedure_call"/>
      <OMSTR>Evaluate</OMSTR>
    </OMA>
    <OMFOREIGN encoding="MathML-Presentation">
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <mrow>
          <mi>sin</mi>
          <mo>&ApplyFunction;</mo>
          <mfenced><mi>x</mi></mfenced>
        </mrow>
      </math>
    </OMFOREIGN>
  </OMA>
</OMATTR>
</OMOBJ>

```

B.2 An example of the `procedure_completed` message

The procedure `GroupIdentificationService` from the previous example returns its successful output in the following form:

```
<OMOBJ>
  <OMATTR>
    <OMATP>
      <OMS cd="cascall1" name="call_ID" />
      <OMSTR>alexk_9053</OMSTR>
      <OMS cd="cascall1" name="info_runtime" />
      <!-- The runtime in milliseconds as OM integer -->
      <OMI>1234</OMI>
      <OMS cd="cascall1" name="info_memory" />
      <!-- Memory occupied by CAS in bytes as OM integer -->
      <OMI>134217728</OMI>
    </OMATP>
    <OMA>
      <OMS cd="cascall1" name="procedure_completed" />
      <OMA>
        <OMS cd="linalg2" name="vector"/>
        <OMI> 24</OMI>
        <OMI> 12</OMI>
      </OMA>
    </OMA>
  </OMATTR>
</OMOBJ>
```

In the case when a cookie is requested, the `procedure_completed` message may look as follows (we assume the minimal debugging level with no information about the runtime and memory used):

```
<OMOBJ>
  <OMATTR>
    <OMATP>
      <OMS cd="cascall1" name="call_ID" />
      <OMSTR>alexk_9053</OMSTR>
    </OMATP>
    <OMA>
      <OMS cd="cascall1" name="procedure_completed">
      <OMR xref="qx196to40CeX" />
    </OMA>
  </OMATTR>
</OMOBJ>
```

### B.3 An example of the `procedure_terminated` message

This is an example how the procedure `GroupIdentificationService` may return an error message if the error arises at the CAS level (we assume the minimal debugging level with no information about the runtime and memory used):

```
<MOBJ>
  <OMATTR>
    <OMATP>
      <OMS cd="cascall1" name="call_ID" />
      <OMSTR>alexk_9053</OMSTR>
    </OMATP>
    <OMA>
      <OMS cd="cascall1" name="procedure_terminated" />
      <OME>
        <OMS cd="cascall1" name="error_CAS"/>
        <OMSTR>Error, the group identification for groups of size\n
          3628800 is not available called from\n
          <function>( <arguments> ) called from read-eval-loop\n
          Entering break read-eval-print loop ... \n
          you can 'quit;' to quit to outer loop, or\n
          you can 'return;' to continue\n
          brk>\n
        </OMSTR>
      </OME>
    </OMA>
  </OMATTR>
</MOBJ>
```

One of standard errors on the SCSCP level may look as follows:

```
<MOBJ>
  <OMATTR>
    <OMATP>
      <OMS cd="cascall1" name="call_ID" />
      <OMSTR>alexk_9053</OMSTR>
    </OMATP>
    <OMA>
      <OMS cd="cascall1" name="procedure_terminated" />
      <OME>
        <OMS cd="cascall1" name="error_memory"/>
        <OMSTR>Exceeded the permitted memory</OMSTR>
      </OME>
    </OMA>
  </OMATTR>
</MOBJ>
```

## 7. APPENDIX C.

**An example of the of WSDL message**

```

<?xml version="1.0" encoding="utf-8" ?>
<description xmlns="http://www.w3.org/2006/01/wsd1"
  targetNamespace="http://www.symbolic-computation.org/wsd1/idgroup"
  xmlns:tns="http://www.symbolic-computation.org/wsd1/idgroup"
  xmlns:idns="http://www.symbolic-computation.org/schemas/idgroup"
  xmlns:wssoap="http://www.w3.org/2006/01/wsd1/soap"
  xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
  xmlns:wsd1x="http://www.w3.org/2006/01/wsd1-extensions">
  <documentation>
    This is an example of the WSDL message describing the Web service
    that accepts permutation group and returns the catalogue number
    of this group in the GAP Small Groups Library. An error will be
    returned if such identification is not possible by various reasons.
  </documentation>

  <types>
    <documentation>
      The 'types' element describes the kinds of messages
      that the service will send and receive
    </documentation>
    <xs:import namespace="http://www.openmath.org/standard/om20-2004-06-30/"
      schemaLocation="openmath2.xsd" />
    <xs:element name="identifyGroup" type="xs:OMOBJ" />
    <xs:element name="identifyGroupCompleted" type="xs:OMOBJ" />
    <xs:element name="identifyGroupTerminated" type="xs:OMOBJ" />
  </types>

  <interface name = "identifyGroupInterface" >
    <documentation>
      The 'interface' element describes *what* abstract functionality
      the Web service provides
    </documentation>
    <fault name = "identifyGroupFault"
      element = "idns:identifyGroupTerminated" />
    <operation name="procIdentifyGroup"
      pattern="http://www.w3.org/2006/01/wsd1/in-out"
      style="http://www.w3.org/2006/01/wsd1/style/iri"
      wsdlx:safe = "true">
      <input messageLabel="In"
        element="idns:identifyGroup" />
      <output messageLabel="Out"
        element="idns:identifyGroupCompleted" />
      <outfault ref="tns:identifyGroupFault"
        messageLabel="Out" />
    </operation>
  </interface>

  <binding name="identifyGroupSOAPBinding"
    interface="tns:identifyGroupInterface"
    type="http://www.w3.org/2006/01/wsd1/soap"
    wssoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP">
    <documentation>
      The 'binding' element describes *how* to access the service
    </documentation>
    <fault ref="tns:identifyGroupFault"

```

```
        wssoap:code="soap:Sender" />
    <operation ref="tns:procIdentifyGroup"
        wssoap:mep="http://www.w3.org/2003/05/soap/mep/soap-response" />
</binding>

<service name="identifyGroupService"
    interface="tns:identifyGroupInterface" >
    <documentation>
        The 'service' element describes *where* to access the service
    </documentation>
    <endpoint name="identifyGroupEndpoint"
        binding="tns:identifyGroupSOAPBinding"
        address="http://www.symbolic-computation.org/wsd1/groupId" />
</service>

</description>
```