A NEW LINGUA FRANCA FOR SYMBOLIC COMPUTATION

Alexander Konovalov
Centre of Interdisciplinary Research in Computational Algebra
University of St Andrews

University of Kaiserslautern
October 26th, 2010
LINGUA FRANCA

a language that is adopted as a common language between speakers whose native languages are different, e.g. a mixture of Italian with French, Greek, Arabic, and Spanish, formerly used in the Levant.
Motivation

Why do we want to combine computer algebra systems?

- Combining capabilities
- Mixing local and remote
- Going parallel
Combining capabilities

Research problems may require combinations of two or more instances of different systems

- GAP and Maple in CHEVIE for handling generic character tables
- Maple and the PVS theorem prover to get more reliable results
- GAP and nauty in GRAPE for fast graph automorphisms
- GAP as a service to be used by constraint solvers for symmetry breaking in search

Less work than adding capabilities to “home” system and you continue to get benefits of the “foreign” system

Unless they will break the interface!
Some mathematical software doesn’t work on Windows
Some requires large (and perhaps changing) databases
Some is still under development and you want to use the latest version
Some you didn’t realise you wanted to use before you left home
Some may only be released as an online service

So far you only get telnet, cut-and-paste and web browsers
Would be nice to combine local and remote computations
Parallel computations

- Individual processors almost stopped getting faster, but they became more numerous.
- Supercomputers with 100,000 cores already exist, and with million cores are envisaged.

- So we need to learn how to exploit multiple processors if we want to be able to solve larger problems in the future.
- We also want to solve large problems today and do not want to wait.
Packages interfacing GAP to specialised stand-alone C software

- p-Quotient, nilpotent quotient, Knuth-Bendix
- Vector enumerator, C meataxe
- nauty

GAP writes input files for the other programs
GAP invokes the other program
The program writes GAP input to a file (possibly aided by translator) and exits
GAP reads and returns result
Works OK, within fairly serious limitations
1999 - today: GAP4

- GAP 4 can interact with other programs while they run
- Improved string and file handling to facilitate this
- Eventually IO package for UNIX standard low level I/O, including network
- (Enhanced) packages for interacting with
  - ACE coset enumerator, p-Quotient, nilpotent quotient, Knuth-Bendix
  - KANT: Alnuth package
  - **Singular**: packages Singular by M. Costantini and W. de Graaf (2003-2006) and RingsForHomalg (also to other systems) by M. Barakat et al. (2009-2010)
  - AtlasRep package
- Works OK, but each interface is an individual challenge to program and may be broken by other system’s upgrades
- SAGE is essentially built around this approach
Common limitations (from GAP experience)

- Interfaces do not work remotely
- Transmission of large or complex objects may be difficult
- To support new CAS, new I/O convertor is needed. It will rely upon the I/O format, may be subject to parsing errors and may be broken by changes in the other CAS
- OpenMath support: not enough deep (i.e. range of CDs and complete syntax/encodings) and wide (i.e. not many CAS)
- Web services: not interactive, just database access
- Shaped to deal with a particular problem (dc)
- May not work in some operating systems
- May be difficult for the end-user to customise
SCIEncce
Symbolic computation infrastructure for Europe
http://www.symbolic-computation.org

5-year Research Infrastructure project supported by the EU Framework VI programme grant RII3-CT-2005-026133.
• The Centre for Interdisciplinary Research in Computational Algebra,
  University of St Andrews, Scotland
• The Dependable Systems Research Group,
  Heriot-Watt University, Edinburgh, Scotland
• Research Institute for Symbolic Computation,
  Linz, Austria
• Ecole Polytechnique,
  Centre National de la Recherche Scientifique, Paris, France
• Computational Mathematics Group,
  Universität Kassel, Germany
• The KANT group,
  Technische Universität Berlin, Germany
• Discrete Algebra and Geometry group,
  Technische Universiteit Eindhoven, Netherlands
• Institute e-Austria Timisoara,
  Romania
• Maplesoft,
  Waterloo, Canada
Composability goals

- To deliver robust, usable and flexible extensions to our systems allowing any system to be easily used as a server or a client for both general and problem-specific services.

- To be open to connections to other clients and servers, representing other CAS as well as special purpose programs

- To offer an extendable framework for combining systems, which:
  - allows both local and remote communication
  - not relies on the input/output format of a particular system
  - optimises average efforts needed from other systems to join
Direct linking CAS to CAS

Linking CAS to other systems

- **Grid-Clients**
  - SymGrid

- **SOAP-Clients**
  - Web-Apps, Java, C#, other CAS, ...

- **GET-Clients**
  - Scripts, other simple apps, ...

- **Humans**
  - Administration

**Middleware**

- **GAP**
- **KANT**
- **MuPAD**
- **Maple**

**More to come**
Remote procedure call protocol for communication between CAS and any other compatible software (another CAS, grid middleware, web-application, etc.)

- Lightweight, sockets-based
- Implemented within systems rather than in wrappers
- Both protocol instructions and data encoded in OpenMath
- Can be used for direct communication between systems
- SCSCP servers and clients can be “translated” into standard Web services servers and clients using Java SCSCP API
- Basis for symbolic computation on clusters and grids
A standard for representing mathematical objects with respect to their semantics

Semantics vs presentation: is \textit{S}_{42} symmetric group of degree 42? the answer to the Ultimate Question of Life, The Universe, and Everything? 1+2+...+42, or anything else?

Instead, the following XML code is clear about its meaning

```xml
<OMOBJ>
  <OMA>
    <OMS cd="permgp2" name="symmetric_group"/>
    <OMI>42</OMI>
  </OMA>
</OMOBJ>
```
the current OpenMath Standard 2.0 is dated June 2004

the worldwide OpenMath activities are coordinated by the OpenMath Society

The motivating idea is to allow their exchange between various programs, storing in databases, publishing on web, etc.

OpenMath has two encodings: XML and binary

Various CAS may support one or both encodings, or support some only in one direction (OpenMath to CAS or vice versa)

Various CAS may support various OpenMath symbols in both directions or only from CAS to OpenMath or vice versa

But of course this does not prevent a CAS to be able to generate a valid OpenMath input for another CAS
Basic objects: integers, floats, strings, byte arrays, variables, symbols

Symbols consist of a name and a reference to a definition in an external document called a content dictionary (CD)

OpenMath objects can be combined recursively in a number of ways using standard OpenMath constructors: application, attribution, binding, error

See http://www.openmath.org for further details
SCSCP protocol messages are defined in OpenMath content dictionaries `scscp1` and `scscp2`.

SCSCP specification defines OpenMath-encoded messages to and from CAS:
- procedure call
- returning result of successfully completed procedure
- returning a signal about procedure termination

- `http://www.symbolic-computation.org/scscp`
SCSCP messages
procedure_call, procedure_completed, procedure_terminated

RPC
identifier
call_id

Standard errors
error_runtime,
error_memory,
error_system_specific

Info
info_runtime,
info_memory,
info_message

Remote objects
store_session,
store_persistent,
retrieve, unbind

Options
option_runtime,
option_debuglevel,
option_min_memory,
option_max_memory,
option_return_object,
option_return_cookie,
option_return_nothing

Special procedures
get_allowed_heads,
is_allowed_head,
get_transient_cd,
get_signature,
get_service_description

Special symbols
signature,
service_description,
symbol_set, symbol_set_all,
no_such_transient_cd
Is it limited to functionality/data types for which CDs exist?

- Avoid this by allowing *transient* CDs, which contain symbols specific to that service, obtainable from the server on request.

Encoding may be unreasonably bulky, or encoding costs may be too high for some applications.

- Perfectly OK for services to pass data in some private format encoded in a *private* CD or using OMSTRING, OMBYTES or OMFOREIGN element, if that suits the application.

Both transmission of actual mathematical objects and *references* to them are supported.

Also working on new CDs for efficient representation of some common cases (e.g. matrices over finite fields).
Implementations as on today

- GAP, KANT, MuPAD, Maple
- Even more: Macaulay2 (out of box), TRIP (out of box), Coq (prototype), Magma (wrapper), Mathematica (coming soon), ...
- Java OpenMath and SCSCP API: java.symcomp.org
- A collection of tools and prototypes that were built around this API (WUPSI, ISS, LattViz, SkySym, ...)
- C/C++ API that originated from TRIP SCSCP support
GAP implementation of SCSCP

- SCSCP package by AK and Steve Linton
- Provides both client and server parts
- Uses GAP packages IO, GAPDoc and OpenMath
- Client works under GAP 4.4.12 on any platform
- Server works in GAP 4.4.12 modulo exception and error handling functionality that will be introduced in GAP 4.5 and needed to pass errors to the client
- This is not a problem under GAP 4.4.12 as long as you control the server
User-level functionality

- The service provider installs procedures available as SCSCP services and starts the SCSCP server
- This allows to provide securely public SCSCP services
- The client sends request to the server and gets back result
- This is compatible with any SCSCP-compliant system !!!
- The underlying technology is well-hidden: the end-user may know nothing about OpenMath and SCSCP !!!
- Store/Retrieve procedures allowing to work with remote objects not supported in the native system
- Support of InputOutputTCPStreams, compatible with other kinds of streams in GAP
Configuring SCSCP server

1. Specify e.g. in gap4r4/pkg/scscp/config.g:
   - default InfoLevel, host name, port number etc.

2. Put all what you need in the configuration file (e.g. gap4r4/pkg/scscp/example/myserver.g):
   - loading all necessary packages
   - other required GAP code or its reading from other file(s)
   - installation of SCSCP procedures using
     InstallSCSCPprocedure("NameForClient", InternalName);
   - starting the server with RunSCSCPserver(...)

Control where to listen, whom to answer, what to accept

Start GAP with gap myserver.g or as a daemon using the provided script (output may go on screen or be redirected to a file or to /dev/null)
Simplest example

lines from the server configuration file

```plaintext
... InstallSCSCPprocedure( "WS_Factorial", Factorial );
...
RunSCSCPserver("localhost",26133);
```

The client needs to know the name of the remote procedure, the name of the server and the number of the port

```plaintext
gap> EvaluateBySCSCP( "WS_Factorial", [ 12 ], "localhost", 26133 );
rec( attributes := [ [ "call_id", "localhost:26133:12325:GxjuL0vp" ] ],
    object := 479001600 )
```
GAP contains a database of all groups of order up to 2000, except those of order 1024.

For all orders in the database not divisible by 512, groups can be “looked up” to find the unique isomorphic database group.

One of GAP’s extension packages ANUPQ provides this facility for groups of order 512.

But ANUPQ needs a special binary compiled and is not available for Windows, so we might wish to make this feature available as a service to call from our GAP sessions on Windows clients.
Three approaches to group identification

GAP (slow machine or no small groups library) → group G → list of matrices generating G → group G of order 512 → GAP in UNIX environment - ANUPQ works

CAS which "understands" matrices → group id → GAP in Windows - no ANUPQ package

fast and complete GAP installation

complete GAP installation

Clients

Servers
Group -> group id

- Install GAP standard function IdGroup as remotely available procedure

```gap
InstallSCSCPprocedure( "WS_IdGroup", IdGroup );
```

- The client’s call to this procedure will look like

```gap
EvaluateBySCSCP( "WS_IdGroup", [ G ], "far.far.away.net", 26133 );
```
List of matrices -> group id

Create a function to construct and identify a group generated by these matrices

\[ \text{IdGroupByGenerators} := \text{function( gens )} \]
\[ \text{return IdGroup( Group( gens ) );} \]
\[ \text{end;} \]
\[ \text{InstallSCSCPprocedure( "GroupIdentificationService", IdGroupByGenerators );} \]

The client’s call to this procedure may look like

\[ \text{gap> EvaluateBySCSCP( "GroupIdentificationService", [ [m1,m2,m3] ], } \]
\[ \text{"far.far.away.net", 26133 );} \]

Note that errors will be handled automatically
pc-group of order 512 -> group id

How to encode pc-groups?

There is no CD for pc-groups (and even fp-groups)

Since we’re only expecting GAP clients, however, we can use a GAP-specific representation – the integer given by CodePcGroup.

So our server will offer just one function IdGroup512ByCode which will take this number, reconstruct the group from it and return its ID.
pc-group of order 512 -> group id

Server-side setup

```
gap> LoadPackage("scscp"); LoadPackage("anupq");;
gap> IdGroup512ByCode := function( code )
>   local G, F, H;
>   G := PcGroupCode( code, 512 );
>   F := PqStandardPresentation( G );
>   H := PcGroupFpGroup( F );
>   return IdStandardPresented512Group( H );
> end;;
gap> InstallSCSCPprocedure("IdGroup512", IdGroup512ByCode );
InstallSCSCPprocedure : procedure IdGroup512 installed.
gap> RunSCSCPserver("localhost",26133);
```
pc-group of order 512 -> group id

Client-side wrapper

gap> IdGroup512:=function( G )
>   local code, result;
>   if Size( G ) <> 512 then
>     Error( "|G|<>512\n" );
>   fi;
>   code := CodePcGroup( G );
>   result := EvaluateBySCSCP("IdGroup512ByCode", [ code ],
>                             "scscp.st-and.ac.uk", 26133);
>   return result.object;
> end;;

Client-side usage: as user-friendly as standard call to IdGroup

gap> IdGroup512( DihedralGroup( 512 ) );
[ 512, 2042 ]

gap> IdGroup(DihedralGroup(256));
[ 256, 539 ]
Demo and further remarks

- SCSCP is a standard
- May be implemented in different ways
- And follow conventions used by a particular CAS
- No need to implement the full range of OpenMath CDs
- Instead OpenMath CDs may be classified as required, recommended and optional
- Implement SCSCP once, and your CAS will be able to talk to any SCSCP-compliant system
Parallel computing in GAP

- ParGAP package (requires UNIX environment)
- dc (external coordinating software for a specific research problem)
- Condor (job submission system)
- SymGrid-Par (middleware from the SCiEnce JRAI activity)
- Only the latter is SCSCP-compliant
- What can you do only in GAP, avoiding external binaries as much as possible?
Parallel computing with SCSCP

- Creating multiple processes
- Synchronising them
- Waiting for the first available result
- Implemented in GAP: easy to modify
- ParMap skeleton on top of this
Parallel computations with SCSCP

Master-worker skeleton

```gap
gap> ParListWithSCSCP( List([2..6],n->SymmetricGroup(n)),"WS_IdGroup");
#I  master -> [ "localhost", 26133 ] : SymmetricGroup( [ 1 .. 2 ] )
#I  master -> [ "localhost", 26134 ] : SymmetricGroup( [ 1 .. 3 ] )
#I  [ "localhost", 26133 ] --> master : [ 2, 1 ]
#I  [ "localhost", 26134 ] --> master : [ 6, 1 ]
#I  master -> [ "localhost", 26134 ] : SymmetricGroup( [ 1 .. 5 ] )
#I  [ "localhost", 26133 ] --> master : [ 24, 12 ]
#I  [ "localhost", 26133 ] --> master : [ 720, 763 ]
#I  [ "localhost", 26134 ] --> master : [ 120, 34 ]
[ [ 2, 1 ], [ 6, 1 ], [ 24, 12 ], [ 120, 34 ], [ 720, 763 ] ]
```
## ParGAP vs SCSCP

<table>
<thead>
<tr>
<th></th>
<th>MPI</th>
<th>SCSCP</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Communication layer</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Environment</td>
<td>UNIX</td>
<td>anything where SCSCP client/server works</td>
</tr>
<tr>
<td>Supported CAS</td>
<td>GAP</td>
<td>any SCSCP-compliant</td>
</tr>
<tr>
<td>Heterogeneity</td>
<td>As in MPI</td>
<td>No limits on operating system, architecture, location</td>
</tr>
<tr>
<td>Fault-tolerance</td>
<td>As in MPI</td>
<td>Retrying on another worker</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Adding new worker</td>
</tr>
</tbody>
</table>
Profiling with EdenTV: one master, 8 local workers and 2x8 remote workers

Normalised unit group of a modular group algebra: the result is a group of order $3^{242}$

Computed sequentially: 5 hr 8 min, in parallel: 19 m 31 sec. Speedup 15.92
Further details

- SCSCP specification
- Manuals for corresponding SCSCP-compliant CAS extensions
- “Parallel computations in modular group algebras” by AK+Steve Linton, Proceedings of PASCO 2010 (Grenoble, July 21-23, 2010): case study and tutorial on optimising the parallel performance in our model
Some exemplar applications


More info:
http://www.symbolic-computation.org
http://www.gap-system.org
http://www.cs.st-andrews.ac.uk/~alexk/

We are looking for applications and feedback! Thank you for your attention!