Manchester Computing

APIs for Computational Steering

http://www.realitygrid.org

http://www.sve.man.ac.uk/Research/AtoZ/RealityGrid/

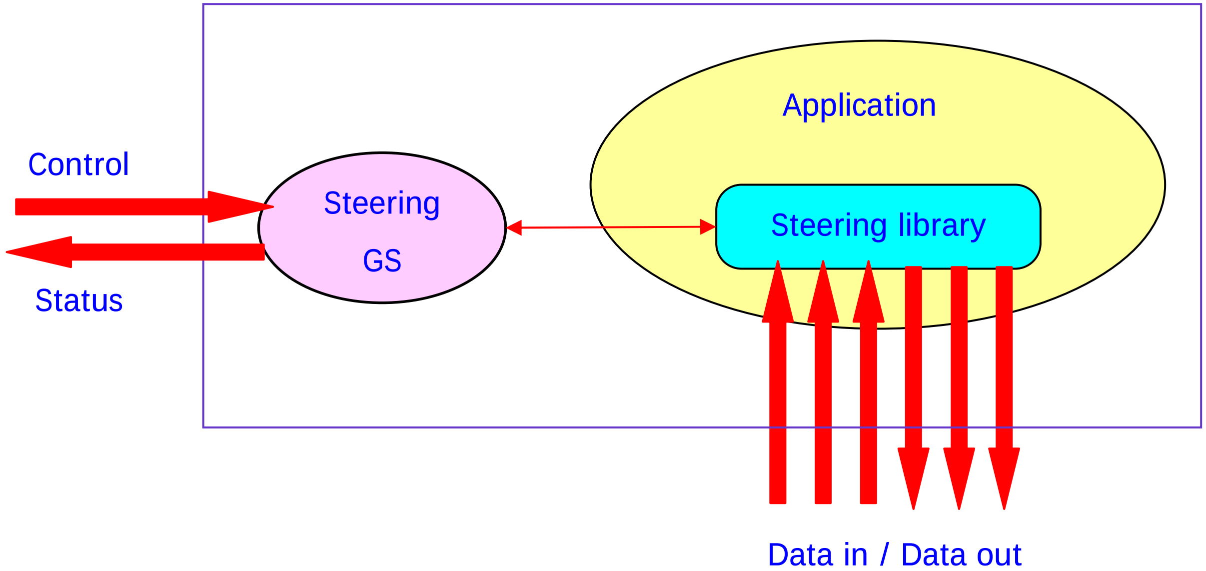
Stephen Pickles

SAGA-RG, GGF 11

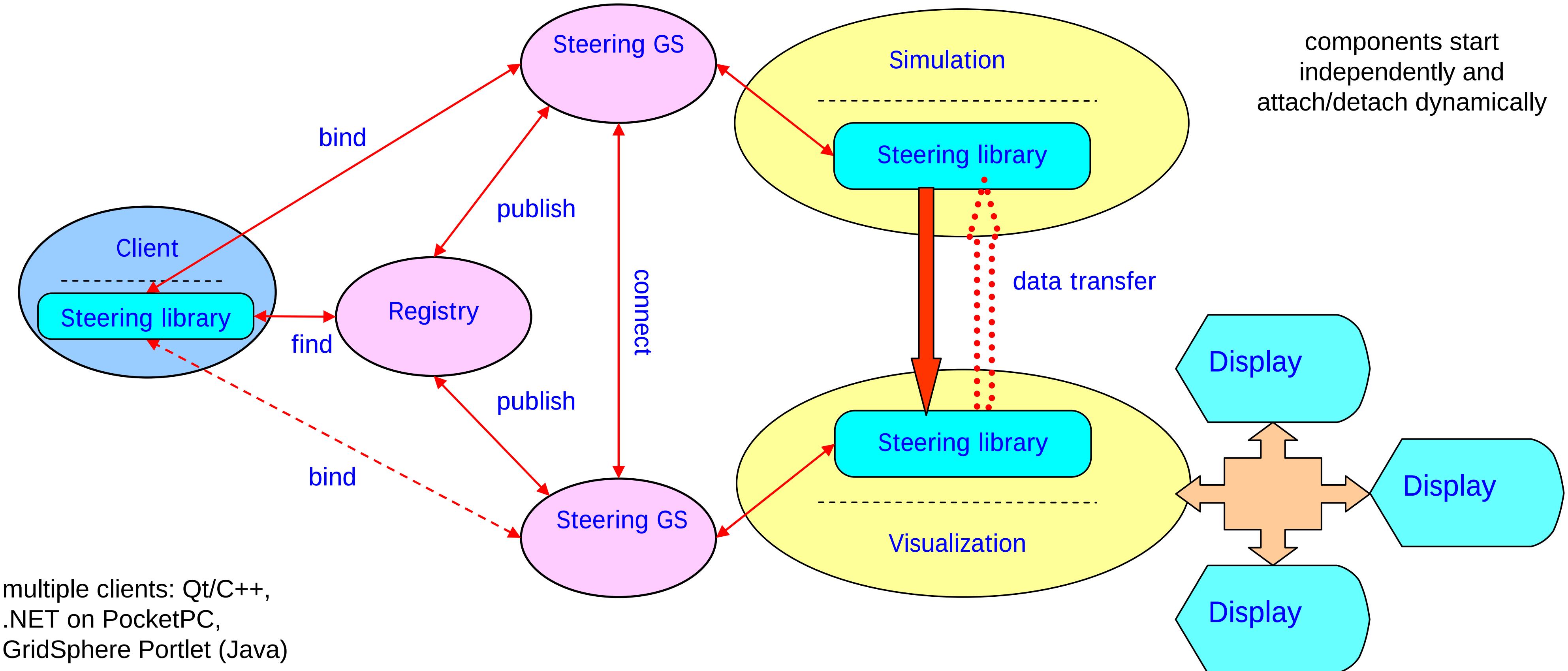
THE UNIVERSITY of MANCHESTER

SAGA and RealityGrid

- Share SAGA philosophy
- Our user interfaces require job submission and file transfer capabilities
  - notice that developers continually wrap lower level commands, eg.
  - Qt launcher shells out to wrapper scripts, which choose between GRAM and ssh
  - writing KIO-Slave for KDE (C++) demands very different APIs to GridFTP
- We also do Computational Steering
  - only approach acceptable to owners of application code is to instrument code for steering through calls to a library


Control
Status
Steering
GS
Application
Steering library
Data in / Data out

Architecture



components start independently and attach/detach dynamically

multiple clients: Qt/C++, .NET on PocketPC, GridSphere Portlet (Java)

Steering operations

- Library provides support for:
  - Pause/Resume and Stop commands
  - Set values of steerable parameters
  - Report values of monitored (read-only) parameters
  - Emit "samples" to remote systems for *e.g*. on-line visualization
  - Consume "samples" from remote systems for *e.g.* resetting boundary conditions
  - Checkpoint and restart
  - Automatic emit/consume with steerable frequency
  - No restrictions on parallelism paradigm
- Bindings in Fortran & C (complete), and Java (client side only)
- You only implement what you need.

Standardisation of Steering?

Opportunities:

- Standardise an API for computational steering
- Standardise the WSDL of the Steering Grid Service

RealityGrid has documented API, library implementations and client tools available for download at:

http://www.sve.man.ac.uk/Research/AtoZ/RealityGrid/

These could be input to a “Simple API”

Questions:

- Is computational steering well understood?
- Is it Simple? Could it be simpler?
- Is there critical mass?

Implementing steering, an example…

An overview of the basic steps required to make a F90 application steerable

Application pre-requisites (1)

- Application code must be written in Fortran90, C, C++ or a mixture of these
- Free to use any parallel-programming paradigm (*e.g.* message passing or shared memory) or harness (*e.g.* MPI, PVM, SHMEM)
- The logical structure within the application must be such that there exists a point (*breakpoint*) within a larger control loop at which it is feasible to insert new functionality intended to:
  - accept a change to one or more of the parameters of the simulation (*steerable parameters*);
  - emit a consistent representation of the current state of both the steerable parameters and other variables (*monitored quantities*);
  - emit a consistent representation of part of the system being simulated that may be required by a downstream component (*e.g*. a visualization system or another simulation).

Application pre-requisites (2)

- It must also be feasible, at the same point in the control loop, to:
  - output a consistent representation of the system (*checkpoint*) containing sufficient information to enable a subsequent *restart* of the simulation from its current state;
  - (in the case that the steered component is itself downstream of another component), to accept a sample emitted by an upstream component.

Initializing the library

```
INTEGER (KIND=REG_SP_KIND) :: status
INTEGER (KIND=REG_SP_KIND) :: num_cmds
INTEGER (KIND=REG_SP_KIND), &
   DIMENSION(REG_INITIAL_NUM_CMDS) :: commands
.
! Enable the steering library
CALL steering_enable_f(reg_true)
.
.
.
! Initialize the library and register which of the built-in
! commands this application supports
num_cmds = 2
commands(1) = REG_STR_STOP
commands(2) = REG_STR_PAUSE

CALL steering_initialize_f(“my_sim v1.0”, num_cmds, &
                           commands, status)
```

RealityGrid

Registering a variable as a steerable parameter

```
CHARACTER(LEN=REG_MAX_STRING_LENGTH) :: param_label
INTEGER (KIND=REG_SP_KIND)           :: param_type
INTEGER (KIND=REG_SP_KIND)           :: param_strbl
INTEGER (KIND=REG_SP_KIND)           :: dum_int
.
.
.
dum_int     = 5
param_label = "test_integer”
param_type  = REG_INT
param_strbl = reg_true ! This parameter is steerable

CALL register_param_f(param_label, param_strbl, &
                      dum_int, param_type, &
                      “”, “”, & ! no lower or upper bound
                      status)
```


THE UNIVERSITY of MANCHESTER

Registering an IOType (for data IO)

```
INTEGER (KIND=REG_SP_KIND)                      :: num_types
CHARACTER(LEN=REG_MAX_STRING_LENGTH), &
   DIMENSION(REG_INITIAL_NUM_IOTYPES)           :: io_labels
INTEGER (KIND=REG_SP_KIND), &
   DIMENSION(REG_INITIAL_NUM_IOTYPES)           :: iotype_handles
INTEGER (KIND=REG_SP_KIND), &
   DIMENSION(REG_INITIAL_NUM_IOTYPES)           :: io_dirn
INTEGER (KIND=REG_SP_KIND), &
   DIMENSION(REG_INITIAL_NUM_IOTYPES)           :: io_freqs
.
.
num_types = 1
io_labels(1) = "VTK_STRUCTURED_POINTS_OUTPUT”
io_dirn(1)  = REG_IO_OUT
io_freqs(1) = 5 ! Automatically (attempt to) output every 5 steps

CALL register_iotypes_f(num_types, io_labels, io_dirn, io_freqs &
                        out_freq, iotype_handles(1), status)
```

Instrumenting the main simulation loop

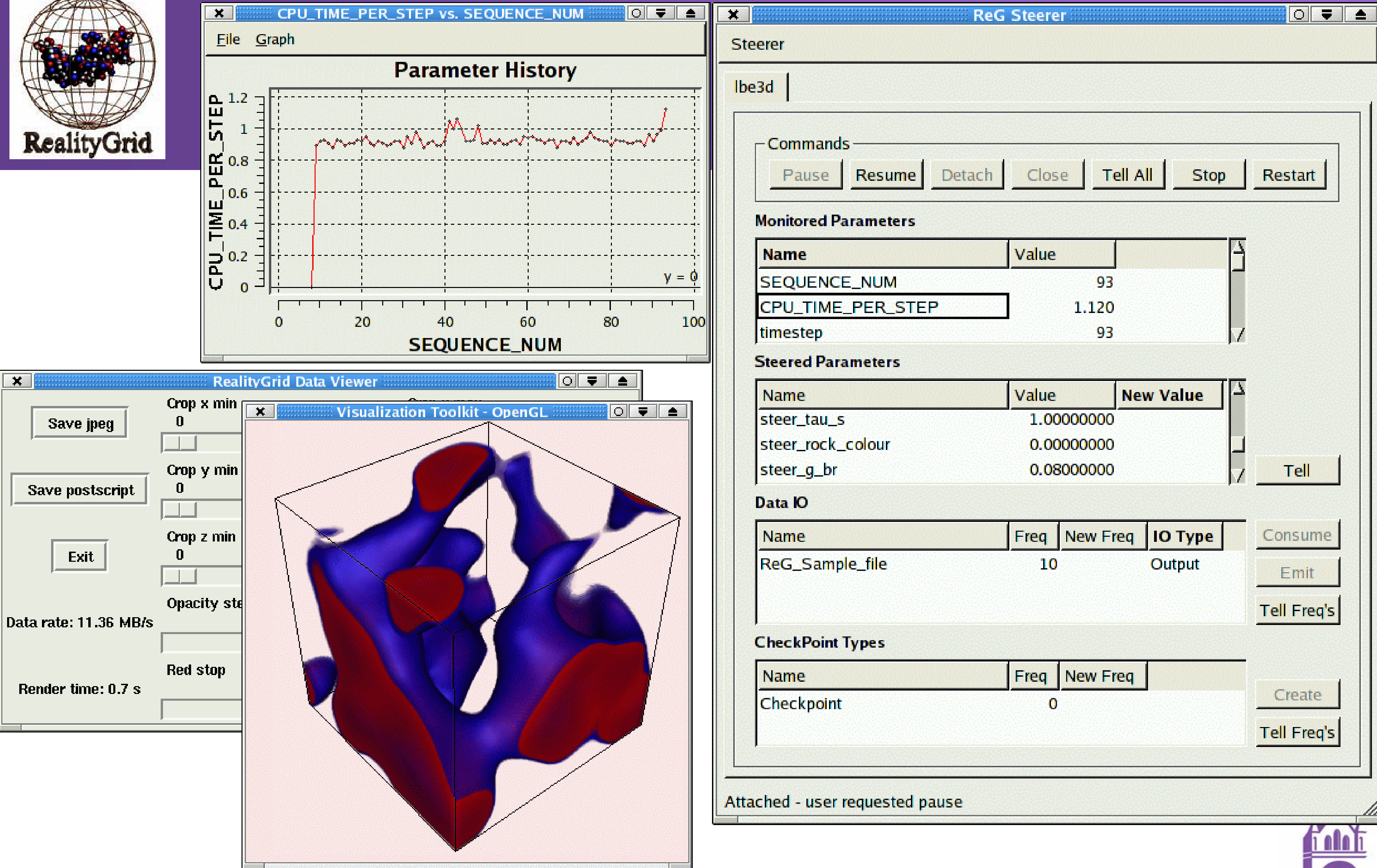
```
! Enter main 'simulation' loop
DO WHILE(iloop<num_sim_loops .AND. (finished .ne. 1))

  IF(my_rank .eq. 0)THEN
    CALL steering_control_f(iloop, num_params_changed, &
                            changed_param_labels, num_recvd_cmds, &
                            recvd_cmds, recvd_cmd_params, status)

    IF(status == REG_SUCCESS .AND. num_params_changed > 0)THEN
      ! Tell other processes about changed parameters here
    END IF
    IF(status == REG_SUCCESS .AND. num_recvd_cmds > 0)THEN
      ! Respond to steering commands here
    END IF
  ELSE
    …
  END IF

  ! Do some science here…
END DO
```

Emitting a data sample

```
! Attempt to start emitting data using an IOType registered previously
CALL emit_start_f(iotype_handles(1), iloop, iohandle, status)

IF(status == REG_SUCCESS)THEN
  ! Send ASCII header to describe data
  data_count = LEN_TRIM(header)
  data_type  = REG_CHAR
  CALL emit_data_slice_f(iohandle, data_type, data_count, &
                         header, status)

  ! Send data
  data_type  = REG_INT
  data_count = NX*NY*NZ;
  CALL emit_data_slice_f(iohandle, data_type, data_count, &
                         i_array, status)

 CALL emit_stop_f(iohandle, status)
END IF
```

Consuming a data sample

```
! 'Open' the channel to consume data
CALL consume_start_f(iotype_handle(1), iohandle, status)

IF( status == REG_SUCCESS )THEN
  ! Data is available to read...get header describing it
  CALL consume_data_slice_header_f(iohandle, data_type, data_count, status)

  DO WHILE ( status == REG_SUCCESS )
    ! Now Read the data itself
    IF( data_type == REG_CHAR )THEN
      ! This assumes c_array is a CHARACTER string of at least data_count chars…
      CALL consume_data_slice_f(iohandle, data_type, data_count, c_array, status)
    ELSE IF( data_type == REG_INT)THEN
      ! This assumes i_aray is an array of integers, at least data_count in length
      CALL consume_data_slice_f(iohandle, data_type, data_count, i_array, status)
    END IF
    ! Get the header of the next slice
    CALL consume_data_slice_header_f(iohandle, data_type, data_count, status)
  END DO
  ! Reached the end of this data set; 'close' the channel
  CALL consume_stop_f(iohandle, status)
END IF
```



CPU_TIME_PER_STEP vs. SEQUENCE_NUM
File
Graph
Parameter History
CPU_TIME_PER_STEP
SEQUENCE_NUM
y = 0
ReG Steerer
Steerer
lbe3d
Commands
Pause
Resume
Detach
Close
Tell All
Stop
Restart
Monitored Parameters
Name
Value
SEQUENCE_NUM 93
CPU_TIME_PER_STEP 1.120
timestep 93
Steered Parameters
New Value
steer_tau_s 1.00000000
steer_rock_colour 0.00000000
steer_g_br 0.08000000
Tell
Data IO
Freq
New Freq
IO Type
ReG_Sample_file 10 Output
Consume
Emit
Tell Freq's
CheckPoint Types
Checkpoint 0
Create
Attached - user requested pause
RealityGrid Data Viewer
Save jpeg
Save postscript
Exit
Crop x min
Crop y min
Crop z min
Data rate: 11.36 MB/s
Render time: 0.7 s
Red stop
Visualization Toolkit - OpenGL

RealityGrid

Summary

- Existing F90/C/C++ codes may be made steerable with relatively little effort
- Amount of steering functionality is related to how much code scientist wishes to write
  - Low barrier to overcome
  - Scientist retains control of their code
- Value-added functionality
  - Automatic emit/consume of samples and checkpoints
  - Checkpoint logging
- Several physics-based simulation codes have been instrumented for steering within the RealityGrid project to date
- Steering library and client available for download from:

  http://www.sve.man.ac.uk/Research/AtoZ/RealityGrid/