Computational Steering in RealityGrid

J. M. Brooke[*], P. V. Coveney[†], J. Harting[†], S. Jha[†], S. M. Pickles[*], R. L. Pinning[*] and A. R. Porter[*].

[*]*Manchester Computing, University of Manchester, Oxford Road, Manchester, M13 9PL.*
[†]*Centre for Computational Science, Christopher Ingold Laboratories, University College London, 20 Gordon Street, London, WC1H 0AJ.*

The RealityGrid project (http://www.realitygrid.org) aims both to enable the discovery of new materials through integrated experiments and to understand the behaviour of physical systems based on the properties of their microscopic components using diverse simulation methods spanning many time and length scales. A central theme of RealityGrid is the facilitation of distributed and collaborative steering of parallel simulation codes and simultaneous on-line, high-end visualisation. In this paper, we review the motivations for computational steering and introduce the RealityGrid steering library and associated software. We then outline the capabilities of the library and describe the service-oriented architecture of the latest implementation, in which the steering controls of the application are exposed through an OGSI-compliant Grid service.

1 Introduction

The RealityGrid project's aim is the facilitation of both computational and experimental studies of complex condensed-matter systems.

The computational studies are typically performed at either the atomistic or meso-scales. Within the project, the former type of simulation is used to study such things as the properties of biomolecules [1] and the nano-indentation of iron [2]. Meso-scale simulations on the other hand are used to study systems involving fluid flow where microscopic inter-particle interactions must be accounted for. Such techniques may, for example, be used in the study of the miscibility of a two-component fluid system [3].

On the experimental side, RealityGrid is involved in two areas: the use of the X-ray micro-tomography (XMT) technique (e.g. for the study of the internal structure of porous rocks) and the London Universities' Search Instrument (LUSI). The latter is a robotic system designed to aid in the search for novel ceramic materials.

In all of these cases, physicists must use expensive, specialised resources (e.g. supercomputer or synchrotron) to do their work and it is therefore important that optimal use be made of them. RealityGrid aims to aid in this process and enhance productivity by improving the physicist's ability to interact with his/her experiment or calculation. In this paper we focus on the use of computational steering to achieve this.

2 Computational Steering

Traditionally, large, compute-intensive simulations are run non-interactively. A text file describing the initial conditions and parameters for the course of a simulation is prepared, and then the simulation is submitted to a batch queue, to wait until there are enough resources available to run the simulation. The simulation runs entirely according to the prepared input file, and outputs the results to disk for the user to examine later.

This technique is suitable for some forms of investigation but for others it can lead to a very inefficient use of resources. A solution to this problem is to provide the physicist with a way to interact with his/her simulation while it is running – a process that is called *computational steering*. This may be as simple as allowing the user to monitor the values of some parameters in their simulation and, if necessary, to edit the values of other parameters. However, to aid the physicist in making informed decisions it will frequently be necessary to enable him/her to see a visualisation of some aspect of their simulated system as it evolves. The complexity of this visualisation can be tailored to suit the hardware available to the physicist (high-end workstation, laptop, PDA *etc*.).

3 The RealityGrid Computational-Steering Library and Client

In this section we describe the model used for software applications within RealityGrid and the set of routines with which the physicist can instrument their simulation code for steering.

We also describe the functionality of the steering client that has been built on the steering library.

3.1 Architecture

Within RealityGrid, applications consist of separate software components. In Figure 1 we show a schematic representation of one such application consisting of a simulation component which emits data to a visualisation component.

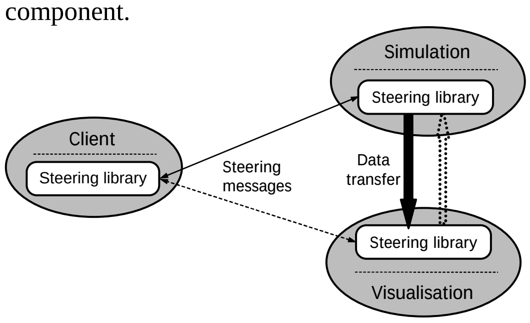

Figure 1: Schematic architecture of simple RealityGrid application.

As indicated in the figure, one or both of these components may be steered using the steering client. This connection can be brought up and taken down dynamically, thus allowing the scientist to periodically examine the state of their simulation while allowing it to run essentially undisturbed for the majority of the time. (The frequency with which the steering library checks for a new connection from a steering client is configurable.)

3.2 The Steering Library

As indicated in Figure 1, RealityGrid components are instrumented for steering by making calls to a library of routines provided for the purpose. The aim throughout has been to enable existing scientific computer programs (often written in FORTRAN90 and designed for multi-processor supercomputers) to be made steerable with a minimum of effort. Minimising the number of changes that an application scientist must make to an existing program is important since it encourages him/her to take responsibility for this work. They then understand the changes that are required and can continue to maintain the software in the future. In light of these requirements, the steering library has been implemented in C which allows it to be used with a variety of common scientific languages such as FORTRAN90, C and C++.

Since different supercomputers/scientists favour different parallel-programming techniques, the library places no restriction on this. The application programmer is free to use MPI, OpenMP *etc*. with the proviso that they take on the responsibility for communicating any changes resulting from steering activity to all processes.

3.2.1 Requirements

In order to make use of the steering library, an application must satisfy certain requirements. In particular, the application must have a logical structure such that there exists a point (which we term a *breakpoint*) within a control loop at which it is possible to carry out the following steering tasks:

i. emit a consistent representation of the state of the application (*i.e.* a set of parameter values);
ii. accept a change to one or more editable parameters;
iii. emit a consistent representation (data sample) of part of the system being simulated (*e.g.* for visualisation);
iv. take a checkpoint or restart from an existing checkpoint.

While all of these things must, theoretically, be possible at the breakpoint, it is up to the scientist as to how many of them his/her application actually supports. For instance, enabling the application to restart from a checkpoint during execution might be a difficult task and therefore need only be attempted if the scientist particularly wants the functionality that that facility will bring.

3.2.2 Functionality

The steering library supports a variety of features. These include the facility for the application to register both monitored (read-only) and steerable (changed only through user interaction) parameters. Beyond this, the library supports a set of pre-defined commands such as 'pause', 'resume', 'detach' and 'stop.' In addition to these, the library allows the user to instruct the application to emit or consume any data sets that it has previously registered. Such actions may also be carried out automatically at a steerable interval. Similarly, the user may instruct the application to take a checkpoint or restart from an existing one. The latter functionality is particularly important since it provides the basis of a system that allows the scientist to 'rewind' a simulation (by restarting from a previous checkpoint). Having done so, it can then be run again, perhaps after having steered some parameter or altered the frequency with which data from the simulation is recorded. The GRASPARC project [4] is an example of another system with this functionality.

In order to maximise the flexibility of the library, we use a system of 'reverse communication' with the application. This means that, for most actions, the library simply notifies the application of what it needs to do. It is then the application's responsibility to carry out the task, possibly using utility routines from the steering library. This is consistent with our philosophy of allowing the scientist to decide how much steering functionality he/she wishes to implement.

3.3 The Steering Client

The steering library consists of two parts – one intended for use by the simulation and the other for client-side applications. Using the latter, a generic steering client has been implemented using C++ and the Qt GUI toolkit [5]. This client may be used to steer any application that has been instrumented using the steering library – the commands supported by an application and its monitored/steered parameters etc. are discovered as part of the connection process. The client GUI is then populated accordingly.

The client can show plots (updated in real time) of the history of one or more monitored parameters. It also provides checkpointing-control, enabling the user to request that the application take a checkpoint (provided it supports such an action) or restart from an existing checkpoint - the user is able to view a snapshot of all parameter values for any logged checkpoint.

4 Steering in the Open Grid Services Infrastructure

We now describe how the RealityGrid steering framework utilises the emerging Open Grid Services Infrastructure (OGSI) [6].

In order to make use of the OGSI, we must represent the software components of Figure 1 as Grid services. However, we again wish to minimise the changes that must be made to existing simulation codes. We have therefore taken the approach, illustrated in Figure 2, where a separate 'Steering Grid Service' (SGS) is used to provide the Grid-service interface of the component. This process communicates with the simulation component via the steering library and thus no code alterations beyond the steering instrumentation are required.

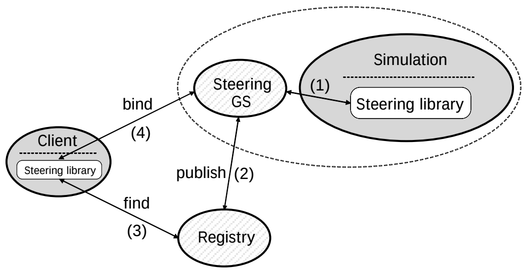

Figure 2: Steering within the OGSI; the numbering indicates the steps in establishing a steering connection.

Within the OGSI, a Grid service is created by a factory service and the SGS is no exception. Thus, the process of launching a simulation job now consists of two steps; a factory service must be asked to generate a SGS and then the simulation itself must be started (using *e.g.* tools from the Globus project [6]) and told the address of its SGS. In principle, the factory could itself also start the simulation but our implementation does not currently support this.

Having established the SGS, it is then possible to make use of standard Grid-service techniques. Thus, the SGS publishes its existence in some registry service (an extension of a ServiceGroupRegistration [7]) which in turn allows a steering client to discover it. Finally, the steering client can now steer the simulation using the methods of the SGS (such as Stop, Pause *etc.*). The communication in steps 2, 3 and 4 in Figure 2 is mediated by SOAP (Simple Object Access Protocol) over http while that in step 1 is private to the implementation of the steering software (but currently also utilises SOAP over http).

The SGS also provides a convenient way of publishing information about the simulation. For instance, in order to establish a sockets-based connection between the visualisation and simulation components of Figure 1, the visualisation (or its SGS) can discover the IP address and port it should use by interrogating the service data of the simulation's SGS.

Currently, the SGS is implemented in Perl and hosted in the container environment provided by the OGSI::Lite package [8]. Since communication is via SOAP over http, the steering client may be run on the scientist's local machine while the simulation itself runs remotely (and the container/SGS may be on yet another machine).

We have implemented a RealityGrid launching framework for the UK's Level-2-Grid [9]. This framework is based upon Globus 2. Many of the resources on the Level-2 Grid are protected by firewalls which presents a problem for the operation of remote steering. However, system administrators typically open a small range of ports for use by Globus. The main problem then becomes one of controlling the user environment on (possibly many) heterogeneous machines. The ability (provided by the SGS) to dynamically configure the software to use one of the open ports greatly simplifies this process. The use of a container environment is also an advantage since it provides a single, configurable point of contact for (potentially) multiple Grid services.

5 Conclusions and Future Directions

We have presented the case for computational steering as a way of improving the efficiency with which valuable computational and experimental resources are used. The RealityGrid implementation of a computational-steering library and client and the use made of the OGSI has been described.

Currently, the OGSI specification does not cover security and at the present time it is unclear whether Grid-service security will be based on (web-services) WS-security or on some other framework. This is an area of ongoing work.

As yet, our implementation does not fully exploit the potential for dynamic service discovery that the OGSI offers, particularly as regards job launching. Ultimately, we aim to have an infrastructure that allows a user to browse the available services and discover information about them (with suitable authentication and authorization). Such functionality will become increasingly important for collaborative working in order to minimise the amount of information that must be explicitly shared between a team of collaborators.

The steering client described in this paper provides basic checkpoint-management functionality. However, it is becoming clear that such functionality is at the heart of many scientific investigative processes. A more sophisticated approach for the storage, retrieval and visualisation of (the availability of) checkpoint data will be developed.

Work is currently underway on the development of additional steering clients (a web portal and a .NET client) utilising the SGS interface. The provision of such clients will give the scientist greater freedom in the platform they can use to interact with their simulation.

6 References

[1] S. Jha, P. V. Coveney and C. Naughton. *Forcefield validation using NAMD: comparing energies and dynamics of DNA.* In preparation (2003).

[2] R. Smith, D. Christopher, S. D. Kenny, A. Richter and B. Wolf. *Defect generation and pile-up of atoms during nanoindentation of Fe single crystals.* Phys. Rev. B, **67**, 245405 (2003).

[3] J. Chin, J. Harting, S.Jha, P. V. Coveney, A. R. Porter, and S. M. Pickles. *Steering in computational science: mesoscale modelling and simulation.* Contemporary Physics. Accepted for publication (2003).

[4] K. W. Brodlie, L. A. Brankin, G. A. Banecki, A. Gay, A. Poon and H. Wright. *GRASPARC: A problem solving environment integrating computation and visualization.* In G. M. Nielson and D. Bergeron, editors, *Proceedings of IEEE Visualization 93 Conference*, p. 102. IEEE Computer Society Press, 1993.

[5] Trolltech Qt – A GUI Toolkit, http://www.trolltech.com.

[6] I. Foster and C. Kesselman, *Globus: A toolkit-based grid architecture*. In I. Foster and C. Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*, p 259. Morgan Kaufmann, 1999.

[7] S. Tuecke et al., *Open Grid Services Infrastructure (OGSI) (draft).* OGSI Working Group of the Global Grid Forum, 2003. http://www.gridforum.org/Meetings/ggf7/drafts/draft-ggf-ogsi-gridservice-23_2003-02-17.pdf.

[8] M Mc Keown, *OGSI::Lite – a Perl implementation of an OGSI-compliant Grid Services Container*. http://www.sve.man.ac.uk/Research/AtoZ/ILCT

[9] R. Allan et al., *Building the e-Science Grid in the UK: Middleware, Applications and Tools deployed at Level 2.* In *Proceedings of All Hands Meeting 2003*, Nottingham, UK.