A grid-enabled lightweight computational steering client: a .NET PDA implementation

BY R. S. KALAWSKY, S. P. NEE, I. HOLMES AND P. V. COVENEY

1East Midlands e-Science Centre, Loughborough University, Loughborough, Leicestershire LE11 3TU, UK (r.s.kalawsky@lboro.ac.uk)
2Centre for Computational Science, University College London, 20 Gordon Street, London WC1H 0AJ, UK

The grid has been developed to support large-scale computer simulations in a diverse range of scientific and engineering fields. Consequently, the increasing availability of powerful distributed computing resources is changing how scientists undertake large-scale modelling/simulation. Instead of being limited to local computing resources, scientists are now able to make use of supercomputing facilities around the world. These grid resources comprise specialized distributed three-dimensional visualization environments through to massive computational systems. The scientist usually accesses these resources from reasonably high-end desktop computers. Even though most modern desktop computers are provided with reasonably powerful three-dimensional graphical hardware, not all scientific applications require high-end three-dimensional visualization because the data of interest is essentially numerical or two-dimensional graphical data. For these applications, a much simpler two-dimensional graphical displays can be used. Since large jobs can take many hours to complete the scientist needs access to a technology that will allow them to still monitor and control their job while away from their desks. This paper describes an effective method of monitoring and controlling a set of chained computer simulations by means of a lightweight steering client based on a small personal digital assistant (PDA). The concept of using a PDA to steer a series of computational jobs across a supercomputing resource may seem strange at first but when scientists realize they can use these devices to connect to their computation wherever there is a wireless network (or cellular phone network) the concept becomes very compelling. Apart from providing a much needed easy-to-use interface, the PDA-based steering client has the benefit of freeing the scientist from the desktop. It is during this monitoring stage that the hand-held PDA client is of particular value as it gives the application scientist greater freedom to leave his or her desk but still communicate with their simulation, with the proviso that they remain within the range of a wireless network.

Keywords: human factors; visualization; computational steering; user interaction; grid; personal digital assistants

1. Introduction

The grid has been developed to support large-scale computer simulations in a diverse range of scientific and engineering fields such as condensed matter

One contribution of 27 to a Theme ‘Scientific Grid computing’.
physics, molecular dynamics and computational fluid dynamics (Foster & Kesselman 1999b; Foster et al. 2003). The grid facilitates execution of large-scale computational simulations by harnessing the power of a distributed computing resource (Hey & Trefethen 2003). For extremely large-scale grid systems, such as the US TeraGrid (Catlett 2002), many hundreds of processing nodes (distributed geographically) are typically employed to improve throughput or increase the scale of the simulation. The growing requirement for powerful supercomputing resources is also being driven by the need to process ever more complex models and large datasets (Avery 2002). Scientist’s workflow strategies are also tending towards submission of multiple jobs (running in parallel) to the supercomputing resource. This parallel workflow facilitates comparison between different simulation runs rather than waiting until all jobs have been sequentially processed. An example of where grid-based computing is being undertaken is the RealityGrid project.\(^1\) This project involves computational studies at atomistic and mesoscale levels with the goal of improving productivity by enhancing the scientist’s interaction with the underlying computational simulation. As large-scale grid systems begin to deliver important scientific results there is an urgent requirement to ensure the grid is user-friendly (Chin & Coveney 2004). Some computing jobs used to take weeks or months to complete prior to the introduction of the grid. Within RealityGrid there have been examples where computation across a high performance grid has achieved a dramatic speed increase from 26 days of continuous computing to less than 48 h. For example, a computation to calculate the exact peptide–protein binding energies by steered thermodynamic integration achieved this impressive throughput (Fowler et al. 2004). In this example, the scientist’s workflow involved launching an initial highly scalable classical molecular dynamics application NAMD2 (Kalé et al. 1999) job and then monitoring its progress before spawning ten or more simulations. Each spawned simulation was not quite independent but was derived from a checkpoint part way through the previous simulation. The scientist monitored the convergence or otherwise of the group average as the simulations proceeded. Decisions were taken at various intervals to terminate, extend or launch new jobs. In this application, second or further chained simulations can be launched at the same time as the first, or a short while later. Consequently, even within a 48 h simulation period, the scientist needs to keep an eye on his/her simulation—this means being able to connect to the simulation and check the parameters of interest. Obviously, being tied to the desktop for a long time is a disadvantage so a solution that allows the scientist to operate away from their desk is highly desirable. This paper describes an effective solution to this problem and is based on a lightweight steering client, developed for a personal digital assistant (PDA) with a wireless or cellular network connection.

2. Computational steering: interactive monitoring and control

The first step in allowing the scientist to interact with the simulation is to employ a technique known as computational steering (Mulder et al. 1999). This is an iterative process that requires the scientist to define a series of independent

\(^1\)RealityGrid project: http://realitygrid.org.
variables to be manipulated during the execution of a given simulation. As the simulation proceeds the resulting dependent variables representing the state of the simulation, as well as the results of interest, needs to be displayed. The scientist then steers or fine tunes the computational model while it is running, observing results and changing input parameters. The process is repeated until the desired results are obtained or have reached the point where no further iteration is required (Leech et al. 1996). Consequently, the scientist is an important part of the feedback loop because he/she is in control of the parameter space being searched. Hence, the ability to interactively control and manipulate steering parameters is a major contributor to the usability of the system. Figure 1 illustrates the interactive elements within a typical closed loop computational steering system. Here a manual control method is used where the user initiates requests such as a change in a system control parameter (e.g. temperature) and the system responds directly by running the simulation with the parameter changes in place.

To make this kind of computational steering system more accessible, the computer scientists working within RealityGrid developed a set of steering libraries that can be integrated with a scientist’s application code (Brooke et al. 2003). The RealityGrid steering system allows the user’s simulation to be distributed over the grid architecture as shown in figure 2. An important feature of the steering system is that it supports dynamic connection to the steering services and means that the user can connect and disconnect to their simulation as it executes, allowing the user to run several jobs simultaneously and switch between them. While this technique improves usability there is still an onus on the user to frequently monitor critical simulation parameters to ensure everything is still on track.

Interactive parameter exploration within a simulation can greatly improve a researcher’s understanding of their data relationships through ‘What if?’ scenarios and it is this process that helps reveal otherwise occluded data and associated relationships. Instead of submitting one job the scientist can submit several jobs simultaneously, each with a different set of initial conditions. Consequently, the scientist’s role becomes more demanding because he/she has to monitor a number of jobs. Therefore, until automatic goal-oriented computational steering systems become readily available, steering will remain partially supervised, where the output of the simulations must be checked on a
regular basis for continued valid output. In addition, job submission and resource management systems for computational resources in high demand do not always guarantee a particular time-slot. Invariably, other higher jobs with a higher priority seem to jump the queue. This means the scientist has to connect and disconnect regularly with the simulations in order to inspect the developing results. Unfortunately, this ties the scientist down to the desk and is not very productive, especially if jobs take hours or days to execute.

3. Supercomputing via a PDA: the requirement

A human factors audit was undertaken across the RealityGrid projects in order to capture workflow, detailed user interaction and best practice information (Kalawsky & Nee 2004). Subsequent analysis of the user’s workflow suggested that within many steered applications there was a role for less visualization intensive interaction during the monitoring and control stages of the process. Indeed, some applications work well with simple steering and need only to display numerical and two-dimensional graphical data to allow the user to monitor and control the simulations. Consequently, it was proposed that a lightweight steering client would be beneficial to the scientist who is mainly interested in two-dimensional or numerical data representations as opposed to complex high resolution visualizations. After discussion with the application scientists it was decided that the ideal lightweight steering client would be based on a mobile computing device which could be used wherever there was a connection to the Internet. The authors decided to develop a PDA steering client for the RealityGrid steering library. While PDAs are not particularly well equipped with significant computing and memory resources they nevertheless provide extremely useful connectivity to larger scale systems by means of IEEE 802.11x wireless networks (O’hara & Petrick 1999). One of the motivations behind the use of a PDA as opposed to a laptop computer was that a PDA solution will also work on modern cellular telephones that support Microsoft’s Windows Mobile software platform. This means the scientist can monitor and control their applications wherever they are located. Other solutions are emerging that are based on lightweight Web based portlets, such as the system.
developed by Edinburgh Parallel Computing Centre (Egbert et al. 2004). However, these require more system resources than are available on a PDA. It is also worth noting that a PDA client will also work on laptop and desktop systems, thus offering true ubiquitous access.

4. Design and Implementation of a .NET PDA client

It was important that the PDA steering client was interoperable with the RealityGrid steering libraries and toolkit (available for download; Manchester-Computing 2004) which utilise the emerging Open Grid Services Infrastructure (OGSI; Foster & Kessleman 1999a). In order to make use of the OGSI, the RealityGrid steering system represents software components as grid services. However, it was necessary to be careful to minimize the changes that must be made to existing simulation codes. Figure 3 illustrates how a separate ‘steering grid service’ (SGS) was 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 were required for the scientist’s application in order to implement a PDA-based steering system. The scientists required the PDA steering client to present pictorial graphs of monitored parameters, reporting the parameter values as they changed over time and an ability to update simulation parameters during the runtime of a simulation. The PDA steering system is an example of an OGSI grid service client and as long as the scientist knows or bookmarks a single uniform resource identifier (URI; Duval et al. 2002) on their PDA they can steer any of their simulations. A URI is a globally unique identifier that identifies a Web resource (either a URL or a URN) that is constructed according to the HTTP namespace rules. It was decided that the Microsoft .NET framework should be used to create the PDA steering client because it provides (i) significant built-in functionality through a rich object model, (ii) a variety of ways to interface and integrate with the outside world, (iii) ease of integration with different languages and (iv) easy deployment.

The PDAs used in our system were based on those that could run Microsoft’s Pocket PC2003 operating system. The .NET PDA steering client was developed
using **Visual C#** in the .Net framework because the .Net framework was designed with Web services in mind and web services description language (WSDL) and schema tooling are provided for code generation. **Visual C#** also supports a memory-managed environment and supports a compact .Net framework for PDAs. From a software engineering perspective, the PDA application had to be particularly efficient and consume very little computing and memory resources (this limitation is imposed by the capability of PDA platforms).

5. The OGSI::Lite framework and steering API

The initial PDA steering client was built with OGSI::Lite and the RealityGrid Steering API (Brooke *et al*. 2003). However, the same technique can be applied to other frameworks such as WSRF::Lite (Coveney *et al*. 2004). The WSDL tooling provided by the .NET framework meant we could take the WSDL documents that had been written describing the OGSI::Lite grid services and XML schema, run them through the WSDL.exe tooling in .Net and obtain C# proxy classes that could communicate with OGSI::Lite, all without having to write a single line of SOAP (Simple Object Access Protocol) code. Client interfaces were also created within the .NET environment to support computational steering in OGSI::Lite and other OGSI containers. The OGSI::Lite grid container allows the exposure of the steering library’s elements as grid services allow communication with a running simulation using XML messages that are derived from the OGSI specification, thus passing to the service via SOAP message envelopes (Box *et al*. 2000). The elements of the Steering API that are exposed as OGSI specification grid services (Tuecke *et al*. 2003) are called the steering grid service, or ‘SGS’, and the ‘ServiceGroupRegistration’, or registry service. The SGS is the grid service by which a simulation is monitored and steered while the registry service is the service by which one discovers which simulations are running and able to be steered.

Figure 3 shows how the PDA client links into the RealityGrid Steering API. The steering system had to use SOAP-based XML very efficiently so it could collect, display and graph data obtained from a remote grid service in a timely fashion. Consequently, remote calls had to be asynchronous because it was not possible to guarantee roundtrip times or the success of any call due to possible network issues that could have frozen the user interface while the client waits for a reply. To connect and communicate with the ServiceGroupRegistration and the SGS service it was necessary to create proxy C# classes that produced correctly formed SOAP envelopes. The ‘WSDL.exe’ tool takes a WSDL document and creates proxy C++/C#/VB classes that then call these services. By modifying the URI in the base class it was possible to use the proxy Web service class as a proxy grid service class. Consequently, C# classes were successfully generated using WSDL descriptions of the OGSI::Lite grid services. The data describing the contents of a registry and the simulation parameters in a SGS are then returned in XML. In order to use this data the XML had to be parsed, a process which was facilitated by the .Net framework XML parsing classes. The ease with which a large part of the client application can be automatically generated by using wizards in VC++, VC# and Visual Basic made custom-built interface development possible.
6. Multithreading on the PDA

Originally the simulation parameter values were parsed directly into data structure stored in a user interface object called a ListView using a separate thread (process). Unfortunately, this approach failed to work because the XML parser and the ‘operating system screen update calls’ tried to access the same data at the same time and crashed. Trying to fix this problem by stopping the screen updates while the XML was parsed caused blank screens to appear and was undesirable to the user because the interface locked up and froze. There were essentially two problems. One, the XML parsing took too long accessing data that was being used to draw the screen and, secondly, a synchronization problem existed when adding data to the ListView. The solution to this problem was to implement a mirror data structure (double buffering technique) that was updated using an asynchronous call (a call-back) to the grid service in a separate thread from the user interface thread. Once the data structure parsing was complete, the thread swapped the new and old structures and requested a screen update from the interface thread. This approach had the advantage of allowing any failure of the grid service call (such as failure to connect) to be dealt with by exceptions without affecting the user interface. Multithreading is vital for this type of application because the grid services can stall the interface if they are not asynchronous, or are in another thread from the main/user interface thread. Therefore, it is important to decouple the received data from the displayed data. From a user interface design perspective, the PDA had to present an interface that was easy to navigate because of the possibility of creating a large number of registries containing a large number of simulations. Best-practice user interface guidelines commonly used for Web page design were used to guide the interface design, including one known as ‘The Three Click Rule’ (Hackos 1997). The aim was to allow the user to manipulate or graph any parameter in any job (in any registry) using no more than three interactions of the stylus. During the design of the screen layouts, care was taken to ensure that information on each screen was logical and meaningful, making it clear to the user which application, and which parameters, were being steered from the context of what was displayed on the screen.

7. The PDA steering client: users perspective

The PDA steering client was developed as a portable thin client interface to a SGS that can be used to control and computationally steer compute intensive applications. The current PDA steering client provides four different screens (see figure 4).

The first screen is the registry form (refer to figure 4a). It contains the user’s personal selection of registries. Each registry contains a list of the steerable simulations accessible via that particular registry. The user can simply enter registries by hand or download them from a text file. Since the URIs can be represented as long strings it is easier to place them in a text file and import them into the PDA. The registries are displayed as icons and text, the icons providing a good visual method for allowing the user to quickly distinguish between registries. A registry can be used to organize simulations in any manner the
scientist chooses and need not relate to a specific infrastructure resource, but can be used to group related simulations. Also, registries can be organized by institution, grid resource, project, a personal preference or any other categorization the user wishes. In order to discover which jobs are in a particular registry, the user need only ‘tap’ on the registry of interest and the job form opens automatically, displaying the simulations that are available for computational steering (refer to figure 4b). The interface displays the simulations and their state, such as waiting to start, actively running or paused. This information is indicated by a coloured icon and an associated text string. From a conceptual point of view, it is important to keep in mind that these jobs are not running on the machine containing the registry but on a computer resource elsewhere on the grid. The user may have launched these jobs originally but only needs the URI of his/her registries to gain access and steer them. In order to steer a simulation the user selects the simulation in the job form to automatically open the parameter form (refer to figure 4c). Once the user has reached the parameter form the PDA client steering system then continuously interrogates the SGS that contains the simulation and updates the screen parameters accordingly. Each displayed parameter relates to a parameter that is monitored or can be steered within the simulation. The update rate of the interface is automatically set using information supplied from the simulation and is related to the simulation loop period. The user can easily adjust steerable parameters by selecting an option from a context menu via a ‘right click’ of the interface. The user enters the new value and this is passed to the SGS, which in turn updates the simulation value. Any parameter can be graphed by collecting samples and ‘graphing’ or by watching the graph form building dynamically as it collects samples (refer to figure 4d). The graph form window has dynamically calculated axes.

8. Future enhancements

The .NET PDA client can be used with any application, provided it has been integrated with the RealityGrid Steering libraries. Future work will look at making the PDA functionality available to other applications such as SciRun (Johnson et al. 2002) and Covise. Further development of the PDA client is being undertaken to include a lightweight visualization client that will enable a
simulation’s visualization to be streamed or served to the PDA. While the limited resolution of the PDA screen will never compete with high end visualization systems the authors of this paper are confident that with suitable configuration of the visualization at the remote site a useful visualization via the PDA can be presented. Security is often reported as a concern for wireless networks. However, a very secure protocol known as Wi-Fi Protected Access (Wi-Fi-Alliance 2004) is readily supported by PDA vendors and has been successfully tested with our PDA client. Use of this protocol should alleviate most users’ concerns. Security will be improved by incorporating e-Science certificates for security. PDA technology is also rapidly evolving and this will enable more powerful features (such as interactive visualization with the server) to be achieved.

9. Conclusions

The .NET PDA client has proved to be a very useful addition to the RealityGrid Steering Library and was successfully demonstrated at the e-Science All Hands Meeting in 2004. It was used to steer a number of jobs including a large-scale steering application looking at the exact calculation of peptide–protein binding energies through steered thermodynamic integration (Fowler et al. 2004) running on the computer nodes of the UK National Grid Service and the US TeraGrid. The .NET PDA has been successfully field tested in the home, place of work and elsewhere to control RealityGrid Steering Library-enabled applications. The carefully designed user interface has made it possible to connect very quickly to running simulations and undertake steering within three taps of the stylus from switching the PDA on. As soon as the scientist inputs registry URIs into the .NET PDA steering client, he/she is able to interact and steer the simulations. While this is extremely convenient for the user, probably the most important benefit is the fact that scientists can now access their simulation from anywhere, as long as they have access to a wireless or cellular phone connection to the Internet. We expect the demand for lightweight PDA-based steering clients to grow because they allow the scientist to remain ‘connected’ to their simulations at all times.

The authors wish to acknowledge the support of the application and computer scientists within the RealityGrid project (University College London, University of Manchester, University of Oxford, Loughborough University, Edinburgh Parallel Computing Centre and Imperial College). In particular, the support of Manchester Computing in the form of the OGSI::Lite package and the RealityGrid Steering API is gratefully acknowledged. The assistance of Andrew Porter and Mark McKeown at Manchester Computing was particularly helpful.

References


Manchester-Computing 2004 RealityGrid steering library and toolkit.


Tuecke, S. et al. 2003 Open grid services infrastructure (OGSI) version 1.0. Wi-Fi-Alliance 2004 WPA™ deployment guidelines for public access Wi-Fi® networks. Austin, TX: Wi-Fi Alliance.