

Developing a Roaming PDA-Based Interface for a Steering Client for OGSII::Lite using .Net: Practical Lessons Learned

S. P. Nee and R. S. Kalawsky

Department Of Computer Science
Loughborough University
Loughborough
Leicestershire
UK

Abstract

As part of our contribution to the RealityGrid project we have developed a computational steering client that can be used on a PDA from any wireless access point. This development has the potential to free the application scientist from the desktop as the only means of steering simulations. We have used the PDA steerer to interactively steer compute intensive simulations in OGSII::Lite from a living room, a lecture theatre, whilst on AccessGrid and even from a garden. This paper is a discussion of the practical human computer interaction issues of the PDA client and the practical software engineering issues in developing it in .Net using OGSII::Lite.

1. Introduction

Computational steering is the process by which a physicist can interact with his/her simulation while it is running by guiding the simulation in a direction beneficial to their investigation [1]. The PDA steering client facilitates this process by allowing the physicist to view pictorial graphs of monitored parameters, by reporting the parameter values as they change over time and by providing the ability to update simulation parameters during the runtime of a simulation. The PDA steerer can use any wireless network that has access to the Internet effectively freeing the application scientist from the need to access a desktop computer to control a simulation. It is an example of an OGSII grid service in action and shows that as long as the application scientist knows or bookmarks a single URI (Uniform Resource Identifier) on their PDA they can steer any of their simulations without even needing to log in to a computer. A URI is similar to a web address.

The information gleaned from our development experience is important to pass on, the time savings in generating client interfaces for computational steering in OGSII::Lite and other OGSII containers is considerable when one uses tools such as .Net. The ease with which a large part of client application can be automatically generated using wizards in VC++, VC# and Visual Basic make bespoke interface development possible in short timescales. This

may allow application scientists to write their own interfaces by modifying a generic interface project to suit their needs without the need for specialist help.

It is our opinion that the PDA client steerer is interesting from an e-Science perspective because it provides a good concrete example of the HCI style and level that grid based applications need to attain before their adoption by the wider community of application scientists. Our experience in RealityGrid has been that most grid applications are utilised by the application scientist who is already a “power user” and who has the technical ability to support the computer science research rather than the other way round. Power users may be useful in driving the research forward and as first adopters of our solutions but normal users will determine if grid based applications are adopted en masse. One recent plea from application scientists [2] would seem to support this view and that usability is the central issue for end users.

A picture of the PDA steerer being used on a live system is shown in Figure 1 below. For the rest of this paper screen dumps of the software running on a PDA emulator will be used because of the difficulties in photographing PDA screens.

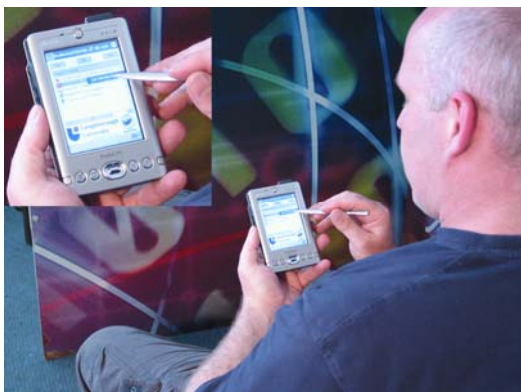


Figure 1 The wireless PDA Steering Client

2 The OGSII::Lite Framework and Steering API.

Our wireless PDA steering client relies on OGSII::Lite [3] and the Steering API [1] developed by researchers at Manchester Computing. The OGSII::Lite grid container allows the exposure of elements of the steering library as grid services and allows us to communicate with a running simulation using XML messages that are derived from the OGSII specification and passed to the service via SOAP message envelopes [5]. The elements of the Steering API that are exposed as OGSII specification grid services [4] are called the steering grid service or ‘SGS’ and the ‘ServiceGroupRegistration’ or registry service. The registry service is the service by which one discovers which simulations are running and steerable and the SGS is the grid service by which a simulation is monitored and steered.

3 The PDA Steering Client

The PDA steering client is a portable thin client interface to a steering grid service that can be used to control and computationally steer compute intensive applications. The current PDA steering client consists of four different screens. The first screen is the registry form (See Figure 2). The registry form contains the user’s personal selection of registries. Each registry contains the steerable simulations accessible via that registry. The user can enter registries by hand or load them in from a text file. Since the URI’s can be long strings it is easiest to put them into a text file and import them. The registries are displayed as icons and text. The icons provide a good visual method for allowing the user to quickly distinguish between registries.

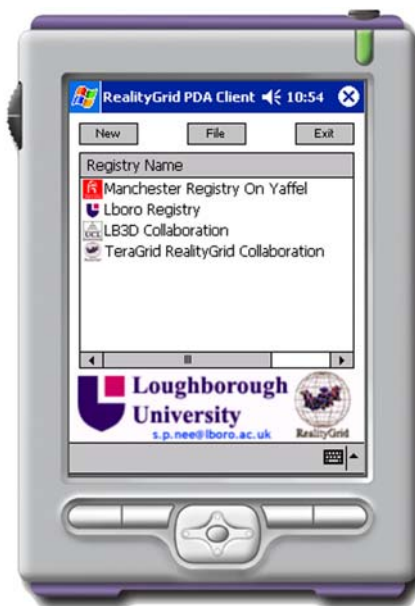


Figure 2 The Registry Form

In RealityGrid a registry can be used to organise simulations in any manner the application scientist wishes, they do not relate to a specific infrastructure resource but can be used to group a related selection of simulations. Registries can be organised by institution, grid resource, project, a personal preference or any other categorisation one could put batches of related simulations into. To discover which jobs are in a particular registry the user clicks on the registry they wish to select and the job form opens automatically displaying the simulations that are available for computational steering (See Figure 3).



Figure 3 The Job Form

The simulations may be waiting to start, actively running or paused, the interface conveys this information by displaying the job icons in different colour icons. It is important to keep in mind that these jobs are not running on the machine containing the registry but on a computer resource 'somewhere' else. The user may have launched these jobs originally but needs only the URI of his/her registries to gain access to them and steer them.

To steer a simulation one selects the simulation in the job form and the parameter form is opened automatically (See Figure 4).



Figure 4 The Parameter Form

Once the user has reached the parameter form the client steerer then continuously interrogates the SGS service that contains the simulation and updates the screen parameters accordingly. Each parameter relates to a parameter that is monitored or steerable within the simulation. The update rate of the interface is set automatically using information supplied from the simulation and is related to the simulation loop period.

The user can adjust steerable parameters by selecting an option from a context menu on the '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. A parameter can be graphed by collecting samples and graphing or by watching the graph form building dynamically as it collects samples (See Figure 5). The graph form window has dynamically calculated axes.



Figure 5 The Graph Form

3 Development Work

3.1 Constraints and Requirements for Prototyping

The development of the PDA steerer was an interesting challenge that posed several software engineering and HCI problems. From a software engineering point of view the application had to be light in footprint because of platform constraints. It also had to use SOAP based XML very efficiently so it could collect, display and graph data collected from a remote grid service in a timely fashion on a platform with very limited CPU power and RAM. The remote calls also had to be asynchronous because we could not guarantee roundtrip times or success of any call due to network issues and could not afford to freeze the user interface whilst the client waits for replies.

From a HCI perspective it had to be easy to navigate because of the possible number of registries containing simulations and possible number of simulations that can be contained within those registries. We adopted a rule of thumb (heuristic) common in web page design, HCI and usability called the "Three Click Rule" to guide our interface development. Our aim was to allow the user to adjust or graph any parameter in any job in any registry using at most three interactions of the stylus, and to make the information on each screen a logical chunk so it was clear where the user was simply by the context of what was on the screen.

We chose to develop using Visual C# in the .Net framework for several reasons.

- .Net framework is designed with web services in mind.
- WSDL and Schema Tooling for free code generation
- Memory managed environment
- Compact .Net framework for PDA
- C# proxies platform independent
 - PDA - Tested
 - Desktop - Tested
 - Portal - TBD

In theory the WSDL tooling meant we could take the WSDL documents that had been written describing the OGSII::Lite grid services and XML schema, run them through the WSDL.exe tooling in .Net and get free C# proxy classes that would communicate with OGSII::Lite without a single line of SOAP code being written and in a matter of seconds.

3.2 Project Creation for a PDA

It was envisaged that the creation of a PDA program would be complicated due to the embedded nature of the device however this was far from the truth. The Visual studio .Net 2003 has a wizard driven PDA project creation device and anyone with limited Windows experience would be able to construct the program. No PDA programming experience is required only minimal Windows, C# and .Net. The authors had no experience of C# prior to this project.

Four windows forms were created based on an initial assessment of the HCI that was needed.

- Registry Form
 - List of registries
- Jobs From
 - List of simulations
- Parameter Form
 - List of steered variables
- Graphing Form
 - Graphing of variable

Into each of these forms the interface was added by dragging and dropping interface elements. The object oriented nature of the C# language and the IDE (Integrated Development Environment) made it very easy to write the code to link them up with events.

3.3 Automatic Code Creation

To connect and communicate with the ServiceGroupRegistration and the SGS service we had to create proxy C# classes that produced correctly formed SOAP envelopes. To create

this code by hand would have been tedious and would have taken a considerable amount of time. Fortunately .Net includes a useful tool called 'WSDL.exe' that will take a WSDL document and create proxy C++/C#/VB classes which can call these services. By modifying the URI in the base class it is possible to use the proxy web service class as a proxy grid service class. The C# classes were generated using WSDL descriptions of the OGSII::Lite grid services and it worked very successfully.

3.4 XML Parsing

The data describing the contents of a registry and describing the simulation parameters in an SGS is returned in XML. In order to use this data the XML must be parsed. The .Net framework incorporates various XML parsing classes that enable this operation to be carried out easily. Parsing XML on a PDA is a relatively slow operation and thus the performance of the application is dependent on this and the roundtrip time for calls to the grid service.

3.5 Multithreading on a 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. This approach failed to work because the XML parser and the operating systems screen update calls tried to access the same data at the same time and crashed. Fixing this by stopping the screen updates whilst the XML was parsed caused blank screens to appear and was undesirable as the whole interface effectively froze. There were essentially two problems; the XML parsing took too long accessing data that was being used to draw the screen and a synchronisation problem existed when adding data to the ListView.

The best solution to this problem turned out to be a mirror data structure 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 method also 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.

Other methods of multithreading were tried but they all had issues related to updating ListView

element. .Net multithreading seemed slightly different to Windows multithreading.

Multithreading is vital for this type of application as the grid services will stall the interface if they are not asynchronous or in another thread from the main/user interface thread. Decoupling the received data from the displayed data is important.

4 Problems and Resolutions

Unfortunately, the WSDL descriptions of the services in OGSII::Lite caused the WSDL.exe tooling crash. After investigation this problem was solved this by using WSDL's generated from a web services project in .Net using only the function prototype definitions of the SGS and ServiceGroupRegistration services. Perl auto-generated WSDL's would seem to be a problem for .Net.

We also discovered that .Net is very reliant on response and request namespaces whilst OGSII::Lite is more forgiving. Once we had the auto-generated C# code running we managed to call the grid services (I.e they registered the call) but nothing seemed to be happening at the client end, the calls just failed. After much investigation and after creating SOAP extensions to capture the raw SOAP envelopes we discovered the problem was unexpected response namespaces returning from OGSII::Lite. .Net gave no warning of this and failed with software exceptions that did not reflect the fault at all but indicated a problem within the XML parsing. This has been solved.

Originally, we tried using the WSDL tooling to create C# classes to receive the XML into C# objects automatically generated from the RealityGrid XML Schema. The intention was to avoid XML parsing altogether and let the operating system do the work. This was a failure as the framework would not work in this manner for us; eventually we gave up and did the parsing using the XML parser within the .Net SDK. We do think that this could be completed at some stage and would make for a much more elegant software design however time was against its inclusion into this project.

5 HCI Issues

The PDA is a small screen device and as such the screen real estate is an issue. Rather than create busy screens that allow the user to do many things we focused on a core set of

functions and used the interface to guide the user through the workflow using a separate screen for each discrete function. This is why we decided to structure the interface so it used four separate windows each obscuring the previous one. The desktop version of this application has an alternative interface that displays all these items in one screen.

The PDA functionality will be increased but at present it is intended only to be used to steer jobs not to launch or destroy them. The ability to destroy a job was included originally but was taken out because it was easy to make a mode error due to the size of the screen and stop a job that might have been running for days in error. Job destruction will be re-implemented as soon as a safe way to do so has been identified.

As can be seen in Figure 4 having the screen font at a readable size means that parameters with long names will not be displayed in full unless the user slides the ListView elements pushing the value of the parameter off screen, this is an issue that will be addressed at some stage. It might be easier to use short names.

Originally the intention was to change the font colour of a parameter to indicate that a value was actively changing so the user's eye would be drawn to dynamic data in visually busy environment. It was not possible to do this in .Net as the only font colour available seemed to be black despite our changing the colour in the code. This code did work on the desktop version however.

The graphing form uses a first generation code for drawing and the screen redraw during updates is obvious and a little annoying. This will be fixed in due course with a better method of drawing the graphs.

When connecting to a registry to obtain the running simulations the interface can appear to hang because it is waiting on a reply from the grid service. This will be fixed in due course by using multithreading, alert boxes and asynchronous grid service calls.

6 Conclusions

Once set up with the registry URI's the PDA steering client allows 24/7 interaction with simulations with no need to login to a computer and it allows the user to gain access to any of their simulation parameters in 3 stylus interactions. The client works and has been

successfully used to steer various high performance parallel codes as well as smaller bespoke test simulations. However, it is yet to be used to do 'real science' but we are involved in projects presently to do this and will report on them soon.

The development process shows that now the WSDL problems have been ironed out it is possible for scientists with access to a little Windows programming experience to write bespoke interfaces that can give them exactly the usability they require. They will not need to worry about networking code at all as the WSDL documents and tooling is present to automatically generate proxy grid service classes in C#, C++ and Visual Basic.

This project was easier than was initially expected. Mainly because of the OGSi::Lite and Steering API being solid and the tooling within .Net helping to generate the SOAP code. The .Net environment had some problems that were overcome however the fact that it failed to notify that it was having difficulties with response namespaces caused about 2 days investigation work.

The project files for the PDA steerer will be added as a demo project to the Steering API as soon as testing has finished and under the same license. This will give a template for other developers to follow should they wish. It is hoped that the code may be a driver that can allow application scientists to try building their own interfaces.

The creation of the PDA steerer now means that RealityGrid simulations can be steered from the desktop, from a web portal and from a wireless handheld device providing full coverage for application scientists who may need it.

7 Future Work

The future work we are intending to carry out includes the launching of simulations via PDA, incorporating e-Science certificates for security and allowing intelligent monitoring agents. We also intend to allow single parameters to be monitored on several jobs so batches of

simulations can be monitored in a single screen. We intend to support WSRF::Lite as well.

7 Acknowledgements

We would like to acknowledge the support of Manchester Computing in the form of the OGSi::Lite package and the Steering API. The assistance of Andrew Porter and Mark McKeown at Manchester Computing was invaluable. The usage of the LB3D code from UCL via Peter Coveney was also very helpful in assessing real world applications.

This work was carried out under grant GR/R67699 from the EPSRC and within the RealityGrid project.

9 References

- [1] J. M. Brooke, P. V. Coveney, J. Harting, S. Jha, S. M. Pickles, R. L. Pinning, A. R. Porter, "Computational Steering in RealityGrid" Proceedings of the UK e-Science All Hands Meeting 2003. Published online <http://www.nesc.ac.uk/events/ahm2003/AHMC/D/pdf/179.pdf>
- [2] J. Chin and P. V. Coveney, "Towards tractable toolkits for the Grid: a plea for lightweight, usable middleware". Published online <http://www.realitygrid.org/lgpaper.html>
- [3] M. Mc Keown, "OGSi::Lite - a Perl implementation of an OGSi-compliant Grid Services Container". Published online <http://www.sve.man.ac.uk/Research/AtoZ/ILCT>
- [4] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maquire, T. Sandholm, D. Snelling, P. Vanderbilt, *Open Grid Services Infrastructure April 5th 2003*. <http://www.gridforum.org/ogsi-wg>
- [5] D.Box, D.Ehnebuske, G.Kakivaya, A.Layman, N.Mendelsohn, H.Frystyk Nielsen, S.Thatte, D.Winer, "Simple Object Access Protocol (SOAP) 1.1" World Wide Web Consortium (W3C) note, May 2000; <http://www.w3.org/TR/SOAP/>