

# Grid-Enabled Desktop Environments: The GRENADE Project

Stephen Pickles<sup>†</sup>, Martyn Foster,<sup>\*</sup> Jon MacLaren,<sup>†</sup> James Marsh,<sup>‡</sup> and Stephen Pettifer<sup>‡</sup>

<sup>†</sup>*Manchester Computing, University of Manchester, Oxford Road, Manchester, M13 9PL.*

<sup>\*</sup>*Silicon Graphics Limited, 1530 Arlington Business Park, Theale, Reading, Berkshire, RG7 4SB*

<sup>‡</sup>*Department of Computer Science, University of Manchester, Oxford Road, Manchester, M13 9PL.*

User interfaces in Grid computing have tended to follow either a command-line or a portal-based approach. Command-line interfaces, such as provided by the Globus Toolkit™, are flexible and powerful, but arcane. Portals deliver their functionality either through Web browsers, or through bespoke, stand-alone “fat clients”; they present friendlier interfaces to a restricted set of services. Despite its astonishing success in making computing accessible to a vast number of users, the desktop paradigm remains strangely neglected by the Grid community. We argue that the command-line and portal approaches have drawbacks which are retarding the uptake of the Grid, and motivate Grid-enabling the desktop itself. Finally, we describe the Grid-Enabled Desktop Environment (GRENADE) project, which is exploring the possibilities that arise when the functionality of the Globus Toolkit is integrated within the user’s desktop environment.

## 1 Introduction

The computational Grid [1] arose from the meeting of advances in distributed computing with genuine requirements in the realms of computational and experimental science. Recognition of the existence of common patterns in Grid computing and business to business integration has brought industrial involvement, adding impetus in the form of the Open Grid Services Architecture (OGSA) [2, 3]. However, building, maintaining and using a production Grid still involve a significant investment of effort [4], which presents a barrier to the widespread uptake of Grid technology. There is little evidence to suggest that the user community is growing at a rate consistent with the hype.

The Grid has been called the Web of the future, but where is the Grid’s “killer app” coming from? How can the Grid deliver its promised ubiquity within the confines of the community that conceived it? Can we enlist the skills and imagination of a wider community to accelerate the development of the Grid? The Data Grid has been called “Napster for scientists”: is it good enough for particle physicists but not friends and family? The Grid should be accessible to ordinary people, not just e-Scientists.

In our vision of the future, there will be robust, interoperable Grid toolkits shipping with every PC; desktops which consume and export Grid services; and thriving open source projects

applying Grid technology to games, file sharing, application sharing, even “friends and family grids”. How do we get there from here?

If the Grid is ever to deliver its promised ubiquity, it must first be made accessible to a much wider community than the one which gave it birth. We believe that an important step towards realising this is to provide improved user interfaces to the Grid.

The remainder of this paper is structured as follows. In section 2, we briefly review current approaches to user interfaces to the Grid, and discuss their strengths and weaknesses. In section 3, we motivate Grid-enabling the user’s desktop environment. Section 4 describes the Grid-Enabled Desktop Environment project (GRENADE) in some detail, and section 5 suggests possible extensions. In section 6, we present our conclusions.

## 2 User interfaces to the Grid

User interfaces to the Grid are the domain of Grid Computing Environments [5]. We identify three main classes of Grid Computing Environments (GCE) in current practice:

1. Command-line interfaces
2. Web portals
3. Stand-alone clients

We will use the term “portal” to refer to the last two collectively.

Command-line interfaces, such as provided by the Globus Toolkit [6], are flexible and powerful, but present a steep learning curve to the average user. Given that the motivation of Globus is to provide a *de facto* standard toolkit on which others can build, it is not surprising that more thought has gone into Globus services and protocols than into user interfaces to them, and some inconsistencies in the various Globus commands persist.

Also worthy of mention is the proposed GCE shell [7, 8]. Here the idea is to view the Grid as a distributed operating system, accessed through a consistent set of user-friendly high-level shell commands, which may in turn be programmed through scripts. In principle — given appropriate separation — such an approach can hide the details of whether the back-end infrastructure is based on Globus, Web services Grid services, and/or something else.

Portals usually deliver their functionality either through a Web browser (Web portals), or through bespoke, stand-alone “fat clients”. They present friendlier interfaces to the user, while restricting access to particular sets of users and services.

Web portals tend to focus on packaging services for user consumption. The kinds of services packaged in this way range widely, from fairly generic job management portals such as CLRC’s HPCGrid Services Portal [9], to more application-specific portals. These share the typical, but not defining, characteristic of a user interface consisting of a series of forms delivered to the user’s Web browser, perhaps supported by Java Applets or browser plug-ins. By routing traffic through the portal, many problems with firewalls can be circumvented; it is easier to arrange to open ports on back-end systems for the portal (whose location is stable) than it is for a host of increasingly mobile client machines.

By leveraging the ubiquity of Web browsers, the portal developer simplifies the problem of Graphical User Interface (GUI) distribution, and is able to expose powerful functionality to the user in a controlled way. However, by choosing what to expose, the developer is equally choosing what to not to expose — thus at the same time as being empowered, the user is also being constrained. The constraints are felt most severely when end users themselves (or third party application developers) want to develop their own interfaces to services on the Grid, perhaps with the aim of linking resources under the control of disjoint portals in complex

workflows of their own devising. If it becomes common practice to hide Grid services behind portals, there is a real danger that power users will be reduced to the expedient of “screen-scraping”, a brittle and unreliable approach. On the positive side, there are encouraging signs that the portals community is recognising the importance of interoperability: enthusiastic acceptance of Web services; early adoption of the OGSA extensions; emerging consensus on the idea of exposing middle tier services; and recent work on interoperable Web portals [10].

The portal approach has many features to recommend it, but uniformity of presentation is not one of them. Although a user might be able to access many portals using the same Web browser, there is nothing to guarantee consistency of style, terminology and behaviour between any two portals.

Stand-alone clients run on the user’s machine, and like Web portals, range from domain-specific problem solving environments to fairly generic job and file management tools. The UNICORE [11] client spans this range by providing a generic workflow management framework that allows the development of application-specific plug-ins.

Even within specific application domains, there is often room for both Web portals and stand-alone clients, and well designed three-tier architectures should cater for both. An example from the UK e-Science programme is computational steering in the RealityGrid project [12]. Here, the Web portal brings advantages to the mobile user and the stand-alone client wins in terms of high-end and remote visualisation.

Both stand-alone clients and Web portals (and to, a lesser extent, command-line interfaces) give rise to, and are liable to perpetuate, a perceptual gulf between “local” and “remote”, the very thing that the Grid should be bridging. As all access to remote resources is ‘through’ the portal, a clear distinction is drawn between the familiar local working environment and the distributed functionality of the Grid. This division complicates the user’s conceptual model — local resources are “on here”, the Grid is very much “out there” — and introduces practical hurdles in terms of interaction between local and remote tools. The sophistication of tools at both ends is often reduced to a lowest common factor by the need for a significant amount of ‘cutting and pasting’ when submitting jobs.

It does not require a great deal of sophistication to publish a Web page and millions of home users do so regularly. On the other hand, a home user offering Grid services to family and friends is unheard of today. We can imagine that Grid technology could be put to good effect in areas such as games, file sharing, and application sharing (license issues aside). But who knows what killer applications might emerge if the imagination of millions of home users could be engaged?

By viewing the user as service consumer, portals tend to neglect the user as service provider. As a consequence, portals offer little or no support for the installation and configuration of middleware, which remains a highly specialised administrative task. Likewise portals tend not to facilitate the incorporation of local resources and applications on the user's own workstation into complex, distributed workflows.

One of the most intriguing characteristics of Grid computing as exemplified by the Globus Toolkit is the lack of distinction between client and server; a system on which Globus is fully deployed is equally capable of using services on other system and providing services of its own. Historically, this has meant that deploying and maintaining Globus has been a non-trivial exercise, with the consequence that more production deployments of Globus are on larger systems and clusters, rather than end-user's PCs. One undesirable side-effect is that Grid users frequently resort to the expedient of distributing copies of their private key around the handful of systems from which they base their Grid activities. For those of us who still dream of a ubiquitous Grid, more disturbing is the way that the dominance of server-style deployments is skewing perceptions of what the Grid can and should become. We would like to see organisations like RedHat bundling toolkits like Globus in their standard distributions because users want them, yet one suspects that it is the cluster market that explains the presence of Globus in the SuSE distribution.

Portals break the symmetry between client and server by focusing on the user as service-consumer, neglecting the service provision capabilities of the client's machine. Command-line interfaces are less restrictive, but even then we find ourselves using different commands to access local and remote resources.

By Grid-enabling the desktop, we preserve the potential of both.

### 3 Grid-Enabled Desktop Environments

The most successful GUI paradigm of our time — the desktop environment — remains neglected by Grid developers. We believe that a serious exploration of the desktop paradigm in the context of Grid computing is long overdue.

A Grid-enabled desktop is a Grid Computing Environment, but it is not a portal, as portals are usually understood. To a first approximation, portals have evolved from the Web server/CGI/browser paradigm, while Grid-enabled desktops represent the evolution of the desktop paradigm (exemplified by Microsoft Windows).

By integrating Grid functionality within the user's desktop environment, we hope to offer more intuitive interfaces without sacrificing the flexibility of the command line.

The Grid-enabled desktop approach helps to reduce the gulf between "local" and "remote". For example, it becomes feasible to allow the user to explore a local disk, a Grid Information System hierarchy or a remote GridFTP-enabled file-system using the same, familiar Graphical User Interfaces.

Moreover, out of a closer coupling between the desktop and the Grid, there arises greater opportunity for integrating personal information resources and processing capabilities on the local machine with the remote resources of the Grid and to harness both in distributed workflows.

### 4 The GRENADE project

The GRENADE project is a partnership between E-Science North West (ESNW) and SGI. The goal of the GRENADE project is to explore the potential of tight integration of Grid functionality within the user's desktop, not to provide "yet another graphical front end".

The project has a two phase strategy:

1. Develop a prototype Grid-enabled desktop environment to pump-prime an on-going open source project,
2. Support the open source project and encourage emulation.

Expected deliverables include single sign-on, Conqueror plug-ins for browsing Grid Information Systems and remote file systems, and GUIs for job definition, submission and management. Users can store shortcuts on their

desktop to their favourite resources, and annotate these with resource-specific preferences.

#### 4.1 Software Stack

GRENADE is built on the Globus Toolkit and the K Desktop Environment (KDE) [13].

KDE [13], the most popular open source desktop, has been ported to a wide variety of operating systems, including Linux, Irix, AIX and Solaris, and ships with most Linux distributions, thus combining a large user base and portability. KDE is layered on Qt [14], a cross platform C++ GUI framework. KDE has an architecture ideal for our purposes, featuring an XML-based framework for componentisation and an elegant, powerful signal/slot mechanism for communication between components. Konqueror, the KDE tool for browsing both local file-systems and the World Wide Web, is extensible — a feature we intend to exploit by providing plug-ins that speak Grid protocols such as GridFTP (for browsing remote file-systems) and GRIP (for browsing Grid Information Systems).

Thus like Web portal developers, we deliver much of our functionality through the browser. However, the motivation is subtly different in the case of GRENADE. Desktop designers have themselves recognised that the power of the browser paradigm is increased when it is used for manipulating local resources (files, programs etc.) as well as for exploring the Web. Thus Explorer is an integral part of the Windows desktop and is much more than just a

Web browser. Similarly, Konqueror is an integral part of KDE, and has subsumed the functions of file management and Web browsing, things that in early versions were delivered through distinct interfaces. The XML-based component model of KDE makes it possible to develop components that can be used equally effectively in stand-alone mode or embedded within Konqueror, while allowing independent components to communicate with each other. This makes possible levels of integration and state management that are impossible or impractical in a portal-based approach.

By using C++ and building on top of the Globus and Qt toolkits, we retain portability and complete access to the Globus commands, scripts and APIs. Although others have found Commodity Grid Kits [15] essential in GCE development, there is no motivation for us to adopt them — the primary goal of CoG Kits is to wrap (or re-implement) Globus functionality so that it can be accessed in other languages. But desktop environments (including KDE and the Qt toolkit on which it is layered) are typically written in C/C++ (not Java, Perl or Python), and the Globus Toolkit itself is written primarily in C.

We have adopted a policy of choosing the Qt classes over their KDE equivalents wherever possible. This is in the interests of portability, and comes at the price of compromising on consistency of look and feel with the KDE desktop — we miss out on KDE themes.

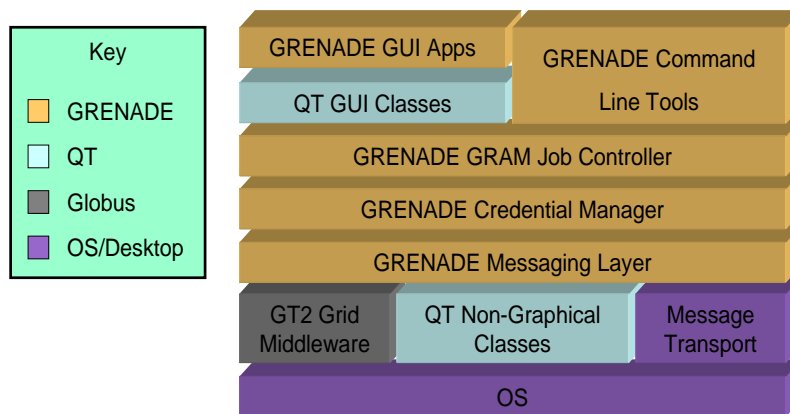


Figure 1. The GRENADE software stack

## 4.2 Architecture

The GRENADE architecture is service oriented, which should simplify future integration with other OGSi-based middleware. GRENADE services, (such as the Credential Manager, and GRAM Job Controller, described below) are embedded in the user's desktop. The GRENADE class hierarchy allows simple, portable and extensible communication between GRENADE applications and GRENADE services.

At the core of the GRENADE architecture (Figure 2) is the GRENADE Messaging Layer. The Credential Manager looks after the user's X509 certificates and keys, and handles the generation of GSI proxies. The GRAM Job Controller looks after jobs launched via GRAM (Globus Resource Allocation Manager).

Both credentials and jobs can be identified by simple names or aliases supplied by the user.

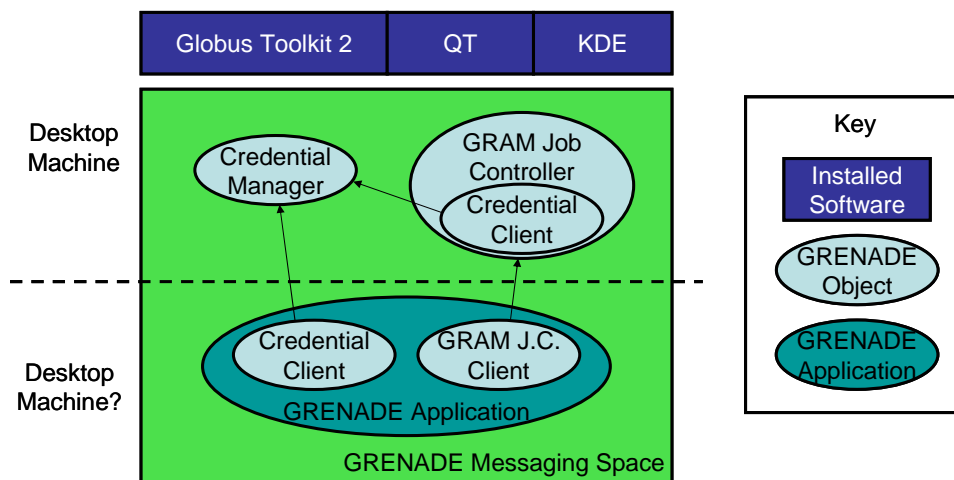


Figure 2. Structure of a GRENADE application

## 4.3 GRENADE Messaging Layer

The GRENADE Messaging Layer is designed so that the messages may be transported over any number of underlying protocols, including KDE's DCOP, Windows messaging, or GSI-enabled SSL. To route over a given protocol, a small handler class must be written for the protocol, and made available for loading at run-time. The messaging layer is able to determine the available transport mechanisms at run-time, so when adding new underlying protocols, there is no requirement to re-link or re-compile parts of GRENADE itself. Similarly any GRENADE application will be immediately capable of using the new underlying protocol.

GRENADE messaging is based on the exchange of serialized C++ Objects. There is a hierarchy of base message classes. Most interactions between an application and a GRENADE service are based around a Remote Procedure Call (RPC) paradigm, i.e. a subclass of `QGrenadeMessageCall` is sent, and a subclass of `QGrenadeMessageReply` is given in return. By convention, all message subclasses begin

“`QGrenadeM`”; also, the reply class for a message `QGrenadeMDoThis` should be `QGrenadeMDoThis_Reply`.

Serialization and de-serialization are much harder to implement nicely in C++ than in Java, due to the nature of the language. As shown in Figure 1, the messaging layer is built on top of the non-graphical Qt classes in our initial prototype. We rely on the limited introspection capabilities provided by the Qt MOC (Meta-Object Compiler) to make serialization and de-serialization work properly. The MOC allows class names and inheritance structures to be deduced at run-time.

### Object References

There is a simple system for referencing a GRENADE object, based upon four separate strings, which are typically expressed colon-separated in a single string. The first string represents the message protocol, e.g. “`dcop`”. The second represents a “message arena”; this string will be interpreted by the support classes for that protocol. For DCOP, the location

includes a machine and a user id; for a different protocol, the location might consist of a machine and a port number. The third string represents a client located in the message arena; this denotes the application, and in the case of DCOP, it is the application name. Finally, the fourth string names the object within the application; a name is a shorthand for the object, assigned when the object is created by, sent to, or registered with, the application.

Of course, as a single application can have multiple interfaces on multiple underlying protocols, therefore a single object may have multiple references. When an object is fetched, it will contain a list of possible references; this allows any client to test if two object references point to the same object.

### Helper Objects

To facilitate the construction of GRENADE objects such as the Credential Manager and GRAM Job Controller, some helper objects are provided as part of the GRENADE Messaging Layer. These include a class which provides a message handler capable of handling RPC-style requests which are farmed out to a number of threads. Such a message processor can be trivially registered with a number of different underlying message protocols.

### 4.4 GRENADE Credential Manager

The GRENADE Credential Manager is an object which is embedded within a user's desktop. The user may give the manager access to certain credentials. The user can use this manager to create and manage GSI proxy certificates. As each credential is given to the Credential Manager, it is assigned a name which can be used to refer to the credential conveniently, e.g. "ukes" might refer to a credential issued by the UK e-Science CA [16], while "globus" might refer to one issued by the Globus CA.

### 4.5 GRENADE GRAM Job Controller

The GRENADE GRAM Job Controller is also embedded in the user's Desktop. It communicates with GRAM JobManagers on the user's behalf. Unlike a GRAM JobManager, the GRENADE GRAM JobController maintains a list of known jobs, allowing the user to ask for a list of their jobs at any given time. This object can carry out most of the basic maintenance operations, such as submission, fetching output, cancellation and so on, and therefore completely

insulates the user from lengthy GRAM Job URLs.

One feature of the Job Controller worth mentioning explicitly is that the Job Controller can set a default credential, either for itself, or for all new client connections. This can optionally be restricted to all new client connections whose client names begin with a given string. This is particularly powerful when used with command line clients, which are connected to the controller for only a short period of time.

## 4.6 GRENADE Applications

A GRENADE application is one built on the GRENADE classes. In addition to the file-system and GIS browser, we are developing applications to control the submission, monitoring and management of GRAM jobs through GRENADE's GRAM Job Controller service. Equipped with both Command Line and Graphical User Interfaces, this GRENADE application adds value to the underlying Grid middleware in a number of ways:

- A simple mechanism for referring to both credentials and jobs by user-supplied names or aliases;
- The ability to list, add, and remove credentials under the control of credential manager;
- Enhanced job management features, including capabilities to list jobs, see how they were started, be notified when a job's status changes, and so on;
- Simplify otherwise awkward things, such as management of multiple credentials, proxy creation in non-standard locations;
- Hide the inconsistencies of existing command line interfaces.

## 4.7 GRENADE Command Line

Among the first classes of GRENADE Applications being developed are command line clients for the GRENADE Credential Manager and Job Controller objects. By convention, GRENADE Command Line client register with their application name prefixed with the name of the machine, user, and terminal. So by using the default settings in the GRENADE Job Controller, the user can then set a default credential for all new connections from their current session (or desktop).

Also, in the GRENADE Command Line, object references are read from right to left, with missing parts of the reference being assigned default values. So, when referring to a proxy credential, the first three parts would be assumed to be “dcop”, “myid@localhost”, “GRENADECredentialManager”; for a GRAM Job reference, the third string would be assumed to be “GRENADEGRAMJobController”.

So to launch a job, a user might type:

```
grenade-list-credentials -type x509
>grenade-create-gsi-proxy -basedcred ukes
```

Here, uk-es is the name assigned to the user’s UK e-Science credential, and results in a proxy with the name ukes-proxy.

Now, set the default credential for the session.

```
grenade-set-default-gram-cred \
-session ukes-proxy
grenade-gram-submit-job \
-name lb3d <parameters>
grenade-gram-list-jobs
```

To sum up, the GRENADE Command Line provides the user with a number of advantages over the GT2 command line.

- Multiple credentials and proxy credentials can be looked up easily
- Credentials can be managed in a consistent way, no matter their location
- Existing GRAM Jobs can be listed

## 4.8 GRENADE Relays

Secure messaging and GRENADE’s service oriented architecture makes it possible to run GRENADE applications remotely, even on systems such as portable devices or Windows PCs. This is because the GRENADE Messaging Layer is independent of Globus and the Credential Manager and Job Controller need not run on every client machine (see Figure 2). Thus a single complete GRENADE installation could act as a gateway, providing Grid capabilities to GRENADE applications running on client systems where GT2 is not, or cannot be, installed.

## 5 Possible Extensions

Although the scope of funded work in the GRENADE project is limited to the deliverables described here, these lay the groundwork for a host of possible future extensions. We have identified a few:

- OGSA integration
- UDDI-based service discovery
- Facilities for management of exported Grid services
- Graphical User Interfaces for replica management
- File-system synchronisation (both local-remote and remote-remote)
- Distributed file system integration (AFS, CXFS, slashgrid)
- Drag-and-drop job submission
- Integration of OpenGL VizServer from SGI
- Support for mobile users.

But we hope that others will be more creative than ourselves, which is why we are committed to supporting GRENADE in the open source well beyond the release of the initial prototype, GRENADE 1.0.

## 6 Conclusions

Others have recognised that the Grid needs better user interfaces, both for its administration and for its everyday use. We have identified shortcomings in current approaches, and have argued that the desktop paradigm deserves serious exploration in the context of Grid computing. We have expressed the view that Grid-enabled desktops have a role to play in increasing the accessibility of the computational Grid, thereby accelerating its uptake and further development, and paving the way towards the genuine ubiquity that the Grid both promises and requires.

We also described the GRENADE project, which is developing a prototype Grid-Enabled Desktop Environment. Using existing Grid and commodity technologies, GRENADE aims to provide tangible enhancements to the desktop environments of the large and sophisticated KDE user base.

## 7 References

- [1] I. Foster, C. Kesselman, S. Tuecke, *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*. International J. Supercomputer Applications, 15(3), 2001.
- [2] I. Foster, C. Kesselman, J. Nick, S. Tuecke, *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*, OGSIG WG, Global Grid Forum, June 22, 2002.
- [3] S. Tuecke et al., *Open Grid Services Infrastructure (OGSI)*, OGSIG WG, Global Grid Forum, 6/27/2003.
- [4] R. Allan et al., *Building the e-Science Grid in the UK: Middleware, Applications and Tools deployed at Level 2*, Proc. AHM 2003 (Nottingham 2-4/9/2003)
- [5] G. C. Fox, D. Gannon, and M. Thomas, "A Summary of Grid Computing Environments". *Concurrency and Computation: Practice and Experience*, Vol 14, No 13-15 (2002).
- [6] Globus project, <http://www.globus.org>
- [7] Geoffrey Fox and Marlon Pierce, *Grid Computing Environments Shells*, proposed GGF Information Document.
- [8] Mehmet Nacar, Marlon Pierce and Geoffrey Fox, *Designing a Grid Computing Environment Shell Engine*, in Proceedings of the 2003 International Conference on Internet Computing, Las Vegas June 2003.
- [9] A.J. Richards, R.J. Allan, G. Drinkwater and K. Kleese van Dam, *Building the e-Science Grid in the UK: An Advanced Grid Programming Environment*, Proc. AHM 2003 (Nottingham 2-4/9/2003)
- [10] Marlon Pierce et al, *Interoperable Web Services for Computational Portals*, SC02, Baltimore, November 2002.
- [11] UNICORE Forum, <http://www.unicore.org>
- [12] 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)
- [13] KDE home page, <http://www.kde.org/>
- [14] Trolltech Qt – A GUI Toolkit, <http://www.trolltech.com>
- [15] Commodity Grid Kits, <http://www-unix.globus.org/cog/>
- [16] UK e-Science Certificate Authority, <http://www.grid-support.ac.uk/ca/ca.htm>