IGR of EPSRC Grant GR/S78346/01 Establishment of an Experimental OGSA Grid

Wolfgang Emmerich
Dept. of Computer Science
University College London
Gower St, London WC1E 6BT, UK
w.emmerich@cs.ucl.ac.uk

John Darlington
London e-Science Centre
Imperial College
180 Queen's Gate, London SW7 2BZ
jd@doc.ic.ac.uk

Malcolm Atkinson, Dave Berry
National e-Science Centre
University of Edinburgh
Edinburgh, EH8 9AA
daveb@nesc.ac.uk

Savas Parastatidis
North East Regional e-Science Centre
University of Newcastle
Newcastle upon Tyne, NE1 7RU
savas.parastatidis@ncl.ac.uk

March 22, 2005

Abstract

At the outset of this one year project, a number of organisations had made initial experience in deploying web services technologies and the Open Grid Services Architecture (OGSA) on campus grids, i.e. within the boundaries of an organisation. The main aim of this project was to establish a testbed that uses web services technologies in general, and implementations of OGSA, in particular, in a truly decentralised manner and across organisational boundaries. The project achieved this objective by establishing an experimental grid based on the Globus Toolkit 3.2 (GT 3.2) implementation that integrated compute resources at UCL, Imperial College, Newcastle and Edinburgh. Once the testbed was established, the project investigated performance, reliability and security of grid service invocation across wide area networks and questions regarding the systematic deployment of grid services into containers hosted on foreign resources. The findings of the project are briefly summarised here and are published in two technical reports available from the project's web site [2, 3].

1 Background

What is OGSA and why did we choose to evaluate it by using GT3.2? At one level, OGSA is just a Service Oriented Architecture (SOA) specifically intended to support Grid applications. OGSA presents a significant departure from the proprietary grid middleware systems, such as Globus 2 and Condor and an orientation to web services standards that are more widely used throughout the computing industry. OGSA builds on these standards and specifies higher-level services that are motivated by, and designed to satisfy, a set of Grid functional and quality requirements.

OGSA Services will be built on core infrastructures (not necessarily one). At the time this project started, the chosen infrastructure was OGSI - the Open Grid Services Infrastructure. We decided to evaluate one implementation of OGSI, GT3.2, as an exemplar, as this was the only real candidate available at the time. Nevertheless, the evaluation and conclusions were designed to inform the development of future approaches to OGSA Grids, either built on GT, or using alternative technologies and products.

The particular objectives of the project included:

- Establishment of a experimental, but secure grid that spans across organisational boundaries and integrates computational resources at UCL, LeSC, NEReSC and NeSC and qualitative evaluation of the effort required to do so.
- Measurement of the performance and scalability that can be achieved when grids are used across wide area networks.
- Measurement of quantitative indicators of the reliability of the grid.
- Investigation into different architectures for the establishment of computational services within the grid.
- Research into systematic deployment of services onto the grid.
- Identification of a list of technical issues that need to be resolved before OGSA-based technology can be deployed in production grids.

2 Key Insights

2.1 Establishment of an Inter-Organisational Testbed

It was agreed in the kick-off meeting of the project not to attempt the resolution of middleware heterogeneity [5] and instead to install the very same version of GT 3.2 on all nodes of the UK OGSA grid testbed. We did agree though to experiment with different hardware and operating system platforms and decided to build the experimental OGSA grid with Intel/Linux, Sparc/Solaris and Intel/Windows machines.

The project succeeded in establishing a working OGSA grid testbed with nodes at UCL, Imperial College, Newcastle and Edinburgh within four months. The testbed configuration evolved from just the basic servlet container with the Axis SOAP engine, to use the Tomcat container and Axis that are currently also in use for the OMII WS-I distribution. In a second phase we then added the higher-level services for GridFTP, registration, notification etc. The grid was eventually secured using the GT3.2 security mechanisms and by using host and personal certificates provided by the UK e-Science Certification Authority. The testbed was then successfully used for performance, scalability and reliability tests. The establishment of the testbed, however, took a considerably larger effort than we originally estimated and was far from simple for a number of reasons.

Prior knowledge of Globus: The experience of the project members at each site with Globus know-how at the start of the project ranged from no prior experience, 3-4 months GT3.0 experience, to three years experience with GT1.X and GT2.X. Some sites had previous experience running inter-organisational grids based on GT2. Because of these differences in platforms and past Globus experience, the issues encountered were not uniform across the test-bed sites. Consequently, a significant amount of project effort was spent trying to determine why something worked easily at some sites but not others.

Research vs. Production Quality: GT3.2 does not claim to be production quality software. It is rather an open source research toolkit, and therefore lacks extensive high-quality support and documentation. In fact, in some ways OGSA is not really intended to be a product at all; it is rather a set of services that are designed to be built upon to solve higher level grid problems. In practice there is only basic built-in tool support for many aspects of the installation, configuration, and management process. It is also very command-line, Linux, and script oriented.

Standards-Compliance: GT3.2 is based on non-standard Web services technology. It allows applications to be exposed as Grid services, and wraps other legacy grid functionality as Grid services. There is a substantial learning curve to master this sort of hybrid technology. Some standard SOAP tools (E.g. JMeter [11], used for load/functional testing) do not automatically work with GWSDL (The Globus specific augmented version of WSDL). The notion of stateful service instances causes problem for tracing/debugging (e.g. standard proxy interception debugging approaches such as TCPMon cannot be used). We also found the choice of development environments that directly support GT3.2 to be small due to this divergence from commercial web services standards.

Administration: Because GT3.2 can be perceived as critical systems infrastructure, with security requirements, its proper administration is significant. Some sites decided to have it installed and managed by the systems administrators, others had dedicated Globus administrators for this role (who then had to interact with systems administrators for specific tasks), while other sites combined both Globus and systems administrator in one role. These choices had significant impact on the time required to install, secure, test and debug, as well as the installation and management approaches that could be chosen. There were a number of site-specific systems administration policies which interacted with GT3 installation. These included the procedures for providing remote users with Globus accounts; the provision of remote access for testing; and authorisation policies.

Platform-Dependencies: GT3 is not fully portable across platforms. It is partly written in Java, and therefore in theory can be easily installed and run on any platform. However, binaries are not always available for all versions, requiring some re-compilation, even of Java code. The bulk of the legacy functionality only runs on UNIX platforms, and it is primarily designed for Linux. Support, testing, and binaries for other UNIXes (e.g. Solaris) was found to be not be as good. Using the right version of supporting software (e.g. Ant, JUnit, C++ compilers, Javac, etc) is also crucial.

Inter-Organisational testing: Traditional tools such as `ping` and TCP tracing/routing programs could not always be used as not all the required ports/protocols are supported end-to-end across multiple organisations and firewalls. Sophisticated commercial tool support (E.g. [1]) is available for this problem, but these tools could not be used because GT3.2 departed from the relevant web service standards.

2.2 Performance, Scalability and Reliability

In order to test the behaviour of the testbed under load, we developed the GTMark benchmark. The benchmark was derived from SciMark2.0 [10], which provided the server side load model. SciMark can be parameterised to run different algorithms, and use more or less CPU cycles and memory. We modified SciMark to provide a suitable load for measuring throughput, and implemented as a GT3.2 Grid Service that is, a callable service which has a unique GWSDL description. The client side work load and metrics gathering capability was based on an Java 2 Enterprise Edition (J2EE) benchmark, Stock-Online [12], which was extended to model the execution of concurrent work-flow instances utilising different tasks.

GTMark is customisable at run time in a number of different ways. The nodes that the benchmark service is deployed on to can be enumerated, or discovered via a directory/index service. Tests can be run for a fixed period of time, or until a steady state for a minimum period of time has been achieved for a specified metric, and can be run with a given concurrency (equivalent to the number of concurrent work-flow instances), or a minimum, maximum and increment to repeat

the test for different concurrencies. It can be configured to measure task throughput, or work-flow response times/throughput (where a complete sequence of work flow tasks must be completed). Fixed, or variable task loads can also be chosen to measure the impact on load balancing of heterogeneous tasks and servers, and the number of tasks per work-flow instance specified. For simplicity (and because we wanted to measure scalability in the simplest possible way) the current version assumes a strict pipeline work-flow, but more complex flows are possible within the same framework (e.g. allowing synchronisation points, and with variable topologies and fan-outs/ins etc).

Performance and Scalability: The impact of security with Tomcat was minimal (unmeasurable) for the two major approaches provided by GT3.2 (encryption and signing). This is somewhat different to what we noticed initially using the GT3.2 container on all sites. It is likely that the GT3.2 container imposes a higher overhead for the security mechanism.

The fastest response time was on the test-bed machine at Edinburgh (3.6s) and the slowest was UCL/Newcastle (26s). As server loads, and therefore response times, increased, client-side timeouts occurred, causing problems as the default client side exception handling naively retried the call – depending on what the problem is this is not always the best approach with stateful services. In fact, calling the same instance a second time was found to kill the container. It immediately started consuming 100% of the CPU time and the service instance could not be deleted. The only cure was to restart the container.

A maximum of 95% of predicted maximum throughput was achieved on four machines. This, along with the slope of the throughput graphs, indicates good scalability. The reason that 100% capacity was not reached could be due to imperfect load-balancing.

In 3 out of 4 cases, the Tomcat and GT3.2 containers gave similar performance. However, on the slowest node (Newcastle), Tomcat was 67% faster than the test container. This was a peculiar result and even after spending considerable time reinstalling parts of the infrastructure, and examining and comparing configurations, JVM settings, etc, no satisfactory explanation was obtained.

On long runs consistently reliable throughput was achieved over a period of hours. For example approx 4,080 calls/hour over 12 hours, delivering a total of 48,000 calls (or 98,000 calls/day).

The following graphs show average response time (ART) in seconds (s) for increasing load (number of client side threads, equivalent to number of server side service instances) (Figure 1 and throughput in calls per minute (CPM) for increasing load (Figure 2).

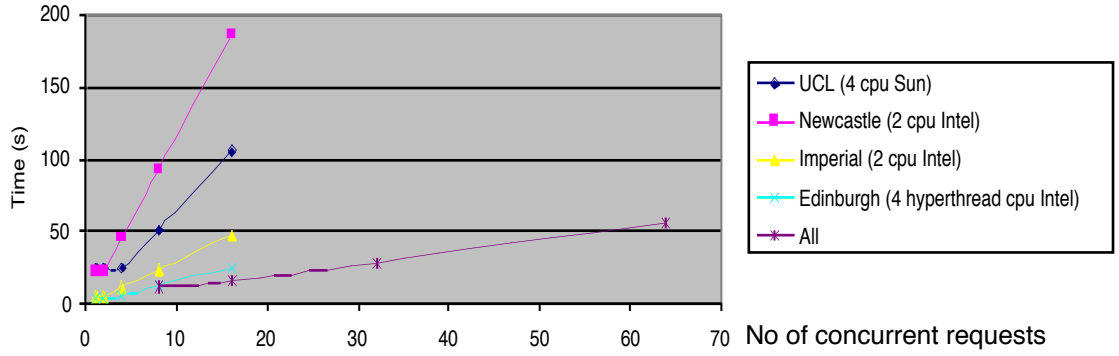

Figure 1: Average Response Time (ART) with increasing load

We also observed that the time taken to create instances using a factory service, and the first call to a new service instance, is slow by comparison to subsequent calls to the same instance (between 1 and 10 seconds slower). This is because there is substantial overhead associated with service creation (e.g. allocating resources), obtaining a grid service handle (GSH) [6], and resolving it to a Grid Service Reference (GSR).

In some tests we noticed that instance creation and destruction was not as scalable as expected. This is possibly because instance creation/destruction is being serialised at some point.

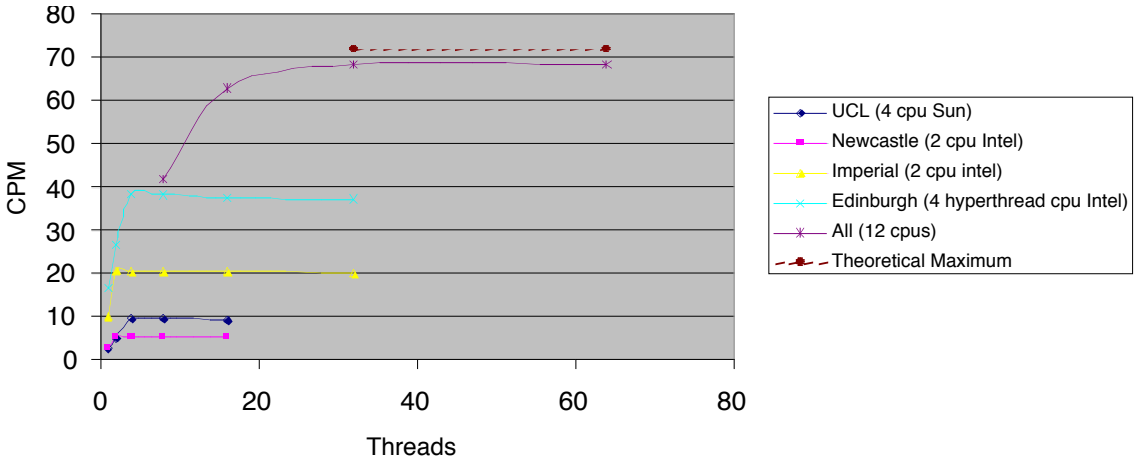

Figure 2: Throughput (CPM) with increasing load

In some tests exceptions were also noted, associated with instance destruction. This issue should be investigated further, as it could cause a problem in highly concurrent applications attempting to initially create a large numbers of service instances, and during the finalisation/cleanup phase when they all need to be destroyed cleanly.

Reliability: During the first round of preliminary tests using the GT3.2 container on all nodes, some runs repeatedly experienced exceptions at a rate of about 1 in 300 calls (a reliability of 99.67%). However, we could not repeat these results with Tomcat installed, and believe that the GT3.2 container was unable to service the incoming HTTP requests adequately. While we believe that GT3.2 with Tomcat now provides high reliability for at least our standard test, we were made aware of multiple possible causes of container failure, and reminded of the problems associated with handling remote exceptions.

For example, it is possible for clients to kill the container, as the container relies on clients being well behaved. A single container may be workable for single users, but with multiple users and applications hosted in the same container the chance of something going wrong increases dramatically.

Containers can be killed by invoking the same method on an instance more than once, for example, after an exception. We assume this is due to a problem with the thread-safeness of stateful instances. Part of the solution is knowing where/when a fault occurred, and what the correct response is. In this case the best approach (assuming no critical state is retained by services across invocations) is to try and kill the instance, and then create another and call it again. If service state is intended to persist across invocations, and if the state has been previously persisted in a database, then it may be possible to kill the instance, and the instance can be recreated and the state re-hydrated.

Containers can also be killed by consuming and saturating their resources. For example, by creating thousands of instances, and invoking them with high concurrency (or even just a few and calling them sequentially if they use sufficient memory). This is relatively easy to do, as numerous objects can be created in a container successfully, but then cause failures once methods on them are invoked due to lazy memory allocation. Eventually the container will saturate (cpu usage), or the memory capacity will be exceeded causing numerous exceptions when services are used or new services created, or even complete container failure. Similar problems have been reported in [9].

Isolation of users, applications, and VOs through the use of more sophisticated security models (e.g. allowing role delegation), and sand-boxing of run-time environments, are likely to be important requirements in production grids. One solution is to isolate users and applications in different containers. This will have scalability limitations eventually, but is a well known enterprise solution (e.g. Borland and Fujitsu J2EE containers can be replicated on the same machine to provide customisable run-time environments, and increase reliability and scalability).

Another approach is to enable containers to manage their own run-time resource usage, by

service instance activation/passivation, so that they do not run out of resources catastrophically. For example, J2EE Entity Beans can be transparently (from the client perspective) passivated/activated at any time, even during a transaction. This mechanism could be used to allow containers to provide a fine-grain (per instance) control over resources, rather than relying entirely on the good behaviour of clients, or external job/resource schedulers, thereby combining policy and finer grained resource handling at the container level. For example, long running jobs could be passivated allowing short-term jobs to use container resources for a short period of time, and then reactivating the previously passivated jobs. Currently some grid job schedulers exclude short-term jobs from gaining resources if long-term jobs are already running (which can cause serious QoS isues, and can even be used to cheat to gain resources in advance of when they are actually needed.

Security and availability are often conflicting requirements. For example, due to problems with the host certificate expiration, notification, and renewal process, and the inability of the process to allow an overlap in the lifetime of certificates, one of the test-bed host machines was unavailable for approximately a week during a critical phase of the test activities. More attention needs to be paid to the management of certificates, in particular the process and infrastructure support for managing the whole lifecycle of certificates.

2.3 Deployment

During the project we elicited deployment requirements and investigated the use of one technology for an automated approach to the deployment of Grid infrastructure and services across organisations.

Within one site: We succeeded in using SmartFrog [7] to fully automate the deployment of GT3.2 at one site. This involved configuring SmartFrog to install and run GT3, Tomcat, and test grid service across multiple machines in the laboratory, and providing a management console to drive the process.

The solution worked well on one site, but relied on the freedom to install and run a completely new installation of GT3 and associated supporting software as an unprivileged user on a public file system. It was also constrained to the deployment of core/container infrastructure only, over a LAN, with no security (either for deployment, or for the GT3 infrastructure). The progress of the installation (along with any exceptions) can be monitored, and a partial (just services) or complete (infrastructure) uninstall also carried out automatically. The deployment process on a single site is scalable and takes no longer for multiple machines compared to one machine.

Across sites: Upon completion of the deployment automation at one site we attempted to deploy GT3.2 across all OGSA test-bed sites using SmartFrog. However, a number of problems were encountered (some expected, others not) including: different security policies at sites; some sites where prepared to open the SmartFrog daemon port with no security, others required security to be enabled, and others may not have been prepared to open an extra port even with security enabled (given the nature of the SmartFrog daemon as an automated deployment infrastructure which can be used to remotely install and run arbitrary code; potentially a perfect virus propagation mechanism).

Secure deployment: The second problem was to get SmartFrog working with security. This was not possible in the available time frame due to issues related to the configuration, use, and debugging of SmartFrog security, and the SmartFrog support group was not able to resolve the security issues in time.

SmartFrog and Globus use different security models and certificates. In order to deploy infrastructure securely with SmartFrog an independent, and largely redundant, security infrastructure, process, and certificates are required, which introduces yet another layer of complexity into already complex infrastructure and security environments. Nevertheless, the SmartFrog security architecture is relatively sophisticated, and includes code signing and multiple trust domains, and is probably well designed for the deployment domain.

Across firewalls on Port 80: In theory RMI over HTTP tunnelling [8] could be used over port 80 to get through fire-walls without having to open extra ports. This may have been harder than it looks to set-up, and would still have required getting SmartFrog security to work to satisfy local security policies.

To deploy secured infrastructure: The next challenge was how to use SmartFrog to install, configure, and run a secure container. In theory all the static security configuration files and host certificates etc can be prepared and then installed using SmartFrog, although there may be some host specific information that needs to be changed. The obvious problem starts when any of the security infrastructure needs to be run as root. The "noroot" option is an alternative for configuring GT3 security without root permission, but we are uncertain how well this works, or if it is appropriate for production environments. A related issue is whether the GT3 infrastructure can be correctly installed and run by the SmartFrog daemon user. It seems probable that the Globus user will be required to install and run it.

Another issue is creating the grid-map files (host and service specific) which map certificates to local user accounts. This requires access to the user certificates for each node, and knowledge of the local user accounts that they are mapped to. It is possible in theory to use a generic single user to run all the jobs for a node, and it may even be the case that for non-mmjfs services a real user account is not needed at all. However, there are significant issues to do with trust, security and auditing if the binding between users and accounts is weakened, although some sort of role-based security mechanism, such as PERMIS [4] is inevitable.

SmartFrog in the laboratory demonstrated the ability to deploy services to a container, and then start the container. However, this assumes that the container is not being shared, and there is only one user (community) per container. The problem with a shared container is that there is *no* hot deployment of services in GT3; a random user cannot just shut the container down whenever someone else is using it (because there is no remote facility to do this, and it would interfere with other users who have services running in the container) – unless there is one container per user (community) or application (which is possible by running multiple containers per server). The extended problem is deploying secured services. Service-specific grid-map files are used, which currently requires knowledge of user certificates and local user accounts.

In the above discussion the assumption has been made that each user can remotely deploy infrastructure and services. In practice this is extremely unlikely, and a confusion of roles. More feasible roles could be: Grid infrastructure deployers, who would deploy infrastructure to a set of resources remotely; and service deployers (for applications or user-community), although they would need to be able to restart containers. This would not be such an issue if services were persistent across container restarts (which is possible in theory with container hosted services as long as the required per-service persistence code has been implemented, but is harder to support with legacy code). How a deployment infrastructure such as SmartFrog can be configured to allow multiple deployment roles (with different capabilities) to share the same deployment infrastructure would need to be considered further. Alternatively the sand-boxing approach of having one container per application or user-community is worth investigating.

We also consider the requirement for a single developer (or user, as in many Grid scenarios scientists both develop and use code) to install/update services across some or all of the grid sites. This is likely as developers are typically in charge of their own code and need to deploy, test, distribute, and update it. This could be done with a configuration management approval mechanism (via the application or user community deployers), or by further delegation of the deployment roles to the user and developer communities. There would need to be mechanisms to prevent conflict in the case of deployment of the same service at the same time, and to ensure service versioning (i.e. that clients use the correct version of services, with backwards compatibility maintained if necessary), and probably audit trails.

3 Recommendations for Further Work

Binary Distribution of Middleware: In order to reduce the installation effort required it would be advisable to make middleware available for a carefully chosen set of hardware and oper-

ating system platforms in binary format. While it may be beneficial to a very few projects to have the source code of middleware available, the large majority of potential grid node administrators will be overwhelmed by the need to compile and install grid middleware for their platform. In many ways it is preferable to have only binary distributions available that are known to work with three or four operating system variants than to have the freedom to operate the middleware anywhere with a result of mediocre installation results. We are pleased to see that this approach is being adopted by the OMII, the ETF for the National Grid Service and EGEE.

Web Service Standards Compliance: Our efforts in producing a testbed were considerably hampered by the departure of GT3.2 from web service industry standards. For example, we were unable to use industry standard remote debugging and performance evaluation tools. Future grid middleware packages need to be much more careful in complying with web service standards and by all means avoiding unnecessary extensions so that the tool suites that may be produced even after the middleware is released will work.

Lightweight Security: Following our experience in establishing testbed security, we extrapolated the cost of security using the current approach of using individual and node certificates. We have estimated the cost for setting up the security for a 100 node grid with 1,000 users to be approximately GBP 300,000. The cost increase is quadratic in the number of nodes and users. We conclude that it will not be economically viable to establish a grid that has more than 1,000 nodes. The detailed estimation is available in [2].

The reason for the cost explosion originates in the requirement to generate an account for each user. Large-scale grids will have to take a different approach. They could for example assume a trust delegation or the role-based access control models that are used in enterprise IT systems. If users were mapped to roles in GT3.2, then the complexity of managing the gridmap files is reduced.

However, authentication is still a problem, requiring user certificates, proxy certificates, and certificate subjects. One further simplification is to allow anonymous users – people who share the same credentials – so that the nodes only have to know about the mapping between classes of anonymous users and roles. However, this allows the possibility for rogue users to misuse their roles without being able to be traced as individuals. Moreover, it is difficult to see how the access rights of individual users could be revoked. Thus further research into the scalability of security models is desperately needed.

Management Console: In order to more easily manage large-scale grids, it is desirable to provide a management console with information about, and control over the following:

- Adding and removing nodes from of a given grid, or VO
- Determining meta-data about nodes such as OS, size/type of machine, and what state they are in.
- Machine and container resources (capacity, history and current usage)
- Support software installed, including the container (version, brand, etc)
- Middleware software installed (services, packages, version, configuration, and status, etc)
- Security, deployment and resource usage information for the container (e.g. what users are authorised for container/services, instance cycling rates, JVM performance statistics, etc)
- Services deployed, deployment information, state, number in use, and resource usage and history
- Access to logs, including informational, event, and error logs
- Start/stop/restart container(s)

Because the roles involved in using and managing a grid infrastructure will frequently cross software and site boundaries, the software infrastructure must be designed to support location independent multiple role scenarios, including remote access to management functions.

Deployment Support: Currently deployment mechanisms for a grid infrastructure is often only added as an afterthought after the infrastructure has been built. Unfortunately, the ability to easily deploy an infrastructure interacts with the security model that was chosen, policies determined by different administrative domains and architectural choices, such as the partitioning of services across containers. As a result deployment requirements need to be considered at the earliest stages of the development, deployment support needs to be reflected in the architecture of the middleware and be appropriately integrated with the security primitives available.

4 Explanation of Expenditure

The expenditure of the project was considerably lower than the grant would have allowed. The grant offer letter would have allowed us to expand the grid with five further nodes after six months. During the course of the project the co-investigators decided that such extension was not advisable for the following reasons:

- The installation effort required for GT3.2 was considerably higher than originally estimated.
- It was deemed unlikely that further insights could be gained by repeating the installation efforts of GT3.2 in more sites.
- We underestimated the effort required to get university administrations to issue sub-contracts to each other and for the team to establish a joined method of working. Issuing five more sub-contracts for three person months and bringing five more RAs on board would have distracted management attention from more important topics.
- By adding further nodes to the network the project's ability to tackle the other objectives might have been jeopardised.
- The infrastructure that was used (OGSI and GT3.2) was discontinued during the course of the project so that there was no additional benefit in extending the knowledge base of these technologies in the UK.

5 Dissemination

The main output of the project were two reports that provide further details of the project's findings. They were published in the technical report series of NeSC and are available from the NeSC web site. Given the extremely short duration of the project it was not possible to publish peer-reviewed articles of the project's results. However papers that summarise the findings are being prepared and will be submitted to peer-reviewed international conferences.

The findings were disseminated in a series of talks as detailed below:

Date	Title	Given at
23/04/2004	UK OGSA Evaluation Project initial findings	E-Science Core Programme Grid and Web Services Town Meeting on Grid and Web Services
15/10/2004	Grid middleware is easy to install, configure, debug and manage - across multiple sites (One can't believe impossible things)"	Oxford University Computing Laboratory
1/11/2004	"Grid Middleware - Principles, Practice, and Potential"	University College London, Computer Science Department
22/03/2005	Lessons learned from the UK OGSA Testbed project	OGSA Testbed Workshop, Westminster University

Detailed findings are also being incorporated into the evaluation process currently driven by Stephen Newhouse (who was a project member) into the ETF middleware evaluation. Moreover, we have had a number of sessions with the Ian Foster and Jennifer Schopf where we have explained the particular findings about GT 3.2 so that ANL can take them into account when building GT4.

Moreover, Stephen Newhouse, Paul Watson and Wolfgang Emmerich serve on the OMII Technical Advisory Board and the experience made during this project is proving valuable for the OMII's effort of building a stable middleware platform based on web service technology.

References

[1] Amberpoint. Exception Manager. www.amberpoint.com/solutions/exception.shtml, 2005.

[2] P. Brebner, W. Emmerich, T. Jones, J. Wu, S. Parastatidis, M. Hewitt, Ol Malham, D. Berry, D. McBride, and S. Newhouse. Evaluation of globus toolkit 3.2 (gt3.2) installation. Technical report, University College London, Dept. of Computer Science, September 2004. http://sse.cs.ucl.ac.uk/UK-OGSA/Report1.pdf.

[3] P. Brebner, W. Emmerich, J. Wu, S. Parastatidis, M. Hewitt, Ol Malham, D. Berry, D. McBride, and S. Newhouse. Evaluating ogsa across organisational boundaries. Technical report, University College London, Dept. of Computer Science, February 2005. http://sse.cs.ucl.ac.uk/UK-OGSA/Report2.pdf.

[4] D. Chadwick. An X.509 Role-based Privilege Management Infrastructure. In *Proc. of the* 17^{th} *Int. Conference on Information Security*. IFIP TC 11, 2002.

[5] W. Emmerich. *Engineering Distributed Objects*. John Wiley & Sons, Chichester, UK, April 2000.

[6] S. Tuecke et al. Open Grid Services Infrastructure (OGSI) Version 1.0. http://www.ggf.org/ogsi-wg, June 2003.

[7] P. Goldsack, J. Guijarro, A. Lain, G. Mecheneau, P. Murray, and P. Toft. SmartFrog: Configuration and Automatic Ignition of Distributed Applications. Technical Report HPOVUA03, HP Labs, Bristol, UK, May 2003.

[8] Sun Microsystems. jGuru: Remote Method Invocation (RMI). http://java.sun.com/developer/onlineTraining/rmi/RMI.html, 2004.

[9] H. Nowell, B. Butchart, D. S. Coombes, S. L. Price, W. Emmerich, and C. R. A. Catlow. Increasing the Scope for Polymorph Prediction using e-Science. In S. Cox, editor, *Proc of the 2004 UK E-Science All Hands Meeting, Nottingham*, pages 968–971. UK Engineering and Physical Science Research Council, 2004.

[10] R. Pozo and B. Miller. SciMark: A Numerical Benchmark for Java and C/C++. http://math.nist.gov/SciMark, 1999.

[11] Apache Jakarta Project. JMeter. http://jakarta.apache.org/jmeter/, 2004.

[12] S. Ran, D. Palmer, P. Brebner, S. Chen, I. Gorton, J. Gosper, L. Hu, A. Liu, and P. Tran. J2EE Technology Performance Evaluation Methodology. Technical report, CSIRO, 2002.