Deployment of Infrastructure and Services in the Open Grid Services Architecture (OGSA)*

Paul Brebner[1], Wolfgang Emmerich[2]

[1] CSIRO ICT Centre, PO Box 664, Canberra, ACT 2601, Australia
Paul.Brebner@csiro.au
http://www.ict.csiro.au/staff/paul.brebner
[2] Department of Computer Science, University College London, London WC1E 6BT, United Kingdom
W.Emmerich@cs.ucl.ac.uk
http://www.cs.ucl.ac.uk/staff/W.Emmerich

Abstract. The ability to deploy Grid infrastructure and services across organizational boundaries (rapidly, reliably, and scalably) is critical for the success of large-scale service based grids such as OGSA. We report the results of the UK-OGSA Evaluation Project infrastructure and services deployment experiments, and analytically compare application versus service deployment. The use of a 3rd party component deployment technology to remotely automate installation and service deployment is discussed, and outstanding problems and potential solutions and benefits are presented. We conclude that grid deployment must be treated as a first-order activity by integrating secure deployment capabilities into the middleware, to enable deployment of secured infrastructure and services across organizations.

1 Grid Deployment Introduction

The UK Open Grid Services Architecture (OGSA) Evaluation [1] was a one year project to establish an experimental OGSA-based [2] grid for the UK E-Science community. The novel focus of the project was to gain insight into issues related to deploying OGSA across organizational boundaries from software engineering and architectural perspectives. Globus Toolkit 3.2 (GT3.2 [3]) was chosen as the exemplar OGSA technology, but the evaluation was designed for the conclusions to be valid for other Service Oriented Architecture (SOA) infrastructures, including GT4. This paper is based on reports and presentations documenting the experimental outcomes of the project [4, 5] and an analytical evaluation of Globus mechanisms [6].

A SOA supports distinct lifecycle steps, namely service development, service deployment, service registration, service discovery and service consumption. In an intra-organizational enterprise context, there are two distinct roles associated with these steps: Provider and Consumer. The provider is responsible for development, deployment and registration of services behind a firewall. The consumer is typically

* This research was funded under EPSRC grant GR/S78346/01

external to the organization, outside the firewall, and discovers and consumes services. The provider role is therefore intra-organizational, while the consumer role is inter-organizational. However, in the Grid community the role division is different. End-user scientists typically develop their own applications, locate resources to run them on, deploy and execute them on these resources, and manage them. The focus is on end-user development, deployment and use, resulting in an overlap of provider and consumer roles, crossing organizational boundaries and firewalls.

GT3 supports end-user deployment of *applications* using mechanisms including: Grid Services, Grid Security Infrastructure (GSI), Master Managed Job Factory Service (MMJFS), Resource Specification Language (RSL-2), data transfer services (Grid File Transfer Protocol - GridFTP, Global Access to Secondary Storage – GASS) and index services (Globus Monitoring and Discovery System - MDS3). For authorized users these allow: job submission services (MMJFS) to be discovered; applications and data to be transferred onto the target machine; resources requested, scheduled and allocated; jobs submitted, run and monitored; and end-users notified upon completion - across organizational boundaries and firewalls.

However, GT3 does not support cross-organizational end-user deployment of grid *services*. This means that an end-user can not easily deploy their science code to resources behind firewalls as a "1st order" grid service. A grid service is an implementation of a Grid Service port type, described (with GWSDL a grid extension of WSDL 1.1), packaged (in a Grid Archive "GAR" file), deployed across organizations to all the containers available to a Virtual Organization (VO), registered in a VO index service, and is discoverable and callable by all the users in the VO who are authorized to use it.

Support for end-user deployment of applications, but not services, reflects two different ways of using a SOA for Grid computing: resource-centric versus service-centric. Scientific applications have typically been large monolithic applications written in legacy languages (e.g. Fortran), for proprietary high performance computing (HPC) architectures. GT3 is designed as a resource-centric infrastructure to allow legacy applications to portably utilize heterogeneous Grid resources. GT3 uses services as *infrastructure* services, to resource science as *applications*. End-users are not expected to deploy new services, but to use pre-existing services to deploy and run their applications. This approach is resource-centric, resourcing *by* services, Web Services enabling of Grids. We define this as a "2nd order" approach, since science is not exposed directly as grid services, only indirectly via infrastructure services. The other approach is service-centric, resourcing *of* services, Grid enabling of Web Services. We define this as a "1st order" approach, since it enables science to be deployed, resourced, and used directly as distinguished grid services.

SOAs exhibit a number of desirable features that are lost if direct execution of science as 1st order services is not supported, including: rich SOA patterns (e.g. proxies for monitoring service use); work-flows for the design and execution of flexible and portable applications and services (through recursive composition); loose coupling of client and service by registration/discovery of services and service descriptions and dynamic binding; and interoperability by conformance to Web Standards.

The original intention of GT3 was also to enhance the *implementation* of Grid services by extending Web Services with a rich component model. In fact, OGSI [7] was explicitly based on J2EE [8]. The OGSI component model includes a set of con-

ventions for service naming and reference, common and extended interfaces and behaviour support for dynamic instance-specific meta-data, and run-time support to manage service lifecycles. Service level security provides a fine-grained security model allowing different levels of access to services, instances and methods. The component model can be used for the development of 1st order science services as well as infrastructure services.

A non-trivial aspect of deployment is Grid infrastructure installation. We consider this to be in the scope of deployment requirements since correct installation and configuration of infrastructure is a precondition for service deployment.

This paper focuses on GT3 infrastructure, application, and service deployment by end-users across organizational boundaries. Sections 2 and 3 compare GT3 installation and deployment for 1st and 2nd order approaches. Scenarios for installation and deployment for each approach are described and an analysis is performed based on the impact of GT3 components and mechanisms on quality attributes. Section 4 details the results of experiments to automate remote installation and deployment, and section 5 reviews related work. Finally, in section 6 we conclude with outstanding problems, potential solutions and benefits of supporting deployment as a 1st order activity in OGSA.

2 Installation of Grid Infrastructure

The grid infrastructure installation scenario steps are as follows: obtain infrastructure components (Globus, and supporting software); discover and select hosts to install to; determine host specific configuration information; install on selected hosts; configure installation on each host; secure installation on each host; start/stop container and services. Related tasks include: Validate installation; discover installation state (what is installed, versions, and configurations); trace installation progress; detect and debug installation failures; reinstall selected components; un-install selected components; install client-side infrastructure and security.

Several months were spent installing GT3 infrastructure across the project's four test-bed sites [4]. The installation experiences varied because of a combination of factors including: platform heterogeneity; site-specific security, and access policies; degree of familiarity with Globus technology; and GT3's fragile build process and complex package structure. Consequently, substantial effort was expended diagnosing and rectifying installation, configuration and access problems, resulting in the following insights.

2.1. Common infrastructure for 1st and 2nd order approaches: Core package, Tomcat container, security, Globus Monitoring and Discovery System (MDS3).

Core package. Installation of the Core package (pure java container related services) and container (e.g. Tomcat) is relatively straightforward, but requires understanding and configuration of Globus and site-specific requirements and policies for installation, access, accounts, and security. On some sites Globus was treated as production

software and installed by systems administrators, entailing extra effort to separate and support roles (for installation, configuration, container management, and deployment). Testing the core installation without security is feasible and is an important step, since it is critical to ensure basic access and functionality before enabling security, as security interferes with remote testing.

Security. Security infrastructure is required for the 2nd order approach, but only for a secured production version of the 1st order approach. In theory the 1st order approach does not require the complete "All Services" package to be installed (the complete middleware stack, which is not pure Java), but some sites installed it for a variety of reasons making security more difficult to install, configure and use than expected [4].

In order to emulate a realistic production grid environment we requested and obtained host and client certificates from the UK e-Science Certification Authority [9]. Accounts were requested and created for users on each site, and client certificate subjects and account mappings configured in "grid-map" files on all nodes. Hosts were configured to use host certificates, client-side security infrastructure was installed and configured, and client-side code was modified to call services with the required security protocol. Unfortunately there is no portable client-side package including security and we were unable to get secured client-side code working under Windows. There are significant problems with certificate management, including the application, acquisition, storing, use, renewing, and revocation processes. A major problem is the lack of scalability of installation and management of security, particularly due to the necessity to provide a unique local account per user, and to map user certificates to local accounts in grid-map files.

It may be possible to run services securely without having individual client accounts on the host machines, making the security process more scalable. It is also not obvious that the GSI approach to security is either sufficient or necessary for a 1st order approach, since GSI was designed for the 2nd order approach and supports proxy certificates, single sign-on, and delegation of credentials. A different security infrastructure may be more appropriate for a 1st order approach, for example, one supporting role based authorization.

Testing and debugging the installation with security enabled is difficult. It is impossible to determine the security configuration of containers and services remotely and to debug calls to secure stateful service instances, since these preclude the use of non-invasive tracing techniques such as proxy interception of calls. It is essential to install infrastructure with tracing and debugging components enabled (E.g. the Axis SOAP handler, SOAPMonitor, and remote Tomcat management; although these have their own security requirements).

MDS3. MDS3 supports a rich index service model, allowing Service Data Elements (SDEs) to be collected, updated, aggregated, cached, persisted and queried for service instances in a variety of configurations. MDS3 was relatively easy to install, configure and test across sites, although it is not part of the core package, many manual configuration and testing steps were required, and configuration errors were not discovered until run-time.

Data transfer. For 1st order services we assume that SOAP attachments are sufficient for data transfer. In practice they are unlikely to be adequate due to bugs in SOAP attachments in Axis/Globus, a practical upper limit to attachment sizes of 1GB,

limited scalability, and incompatibility with security. Otherwise the OGSA data transfer services must also be installed (but are problematic to use with services, see [5]).

2.2 Extra 2nd order infrastructure: "All Services", and data transfer

The 2nd order approach needs the "All Services" package to be installed and configured, including MMJFS (the Job Manager, part of the Web Service Globus Resource Allocation Manager, or GRAM, component), a resource scheduler, and data transfer services. We did not attempt to use a real resource/batch scheduler as Globus does not come with one by default, but used the simpler MMJFS fork instead.

The non-Java packages, and even some of the Java packages, require compilation as part of the installation process. Correct versions and in some cases "brands" of supporting software must be used to guarantee a successful build. Globus is primarily targeted at Linux and support for other flavours of UNIX and other platforms is limited. We experience compilation issues on both Linux and Solaris and the build process was fragile and error prone. Due to version churn and build problems this process had to be repeated frequently, starting from a clean slate each time to eliminate dependencies on previous attempts. Some sites reported issues installing different versions of "All Services" on the same machine.

GridFTP is a legacy Globus component and not well integrated with GT3 services and the container. To support file staging with MMJFS, GridFTP, or a GASS server (for smaller files) must be installed on both server and client machines [10]. Due to problems originating from poor documentation, lack of example code, bugs in the GridFTP server, and certificate issues, we were unable to get data transfer working correctly across sites [5].

2.3 Analysis

Infrastructure weaknesses include portability, build-ability, and packaging. There is a need for well-supported binaries or portable code (i.e. pure Java), better integration and packaging of components, support for adding (or removing) selected components from a working installation, and a portable client-side package (including security). Portability contributes directly to the ability to automate the installation process remotely and therefore scalability of installation. The 1st order approach is intrinsically more portable, since only a container and security are essential, whereas the 2nd order approach relies on legacy non-Java components which are less portable and require more effort to build, install, configure and test before use.

Support for remote viewing of installation state is minimal, with no way to determine what packages and versions have been installed, or how far the installation has proceeded. One obvious problem for security scalability is creating and mapping user certificates to local accounts. Security infrastructure processes and management, including certificates and accounts, need improvement in order to be more useable, scalable and easier to automation. Apart from security, any organization, machine or site-specific configuration makes it more difficult to automate a scalable installation

process. For example, installation location, port number, user access, and site-specific security policies must be successfully negotiated and configured.

We conclude that the infrastructure for 1st order services is amenable to automatic remote installation since it can be better packaged, requires less building, is more portable, requires less configuration, is easier to test incrementally (without security, and then with security), and in theory can utilise a simpler security model. However, improvements in processes, tools and technologies are also required to support: remote automatic installation and configuration; separation of installation roles; monitoring of installation progress and state; visibility of components installed and versions, and security infrastructure; and debugging of installations.

3 Deployment of Services and Applications

In the standard 1st order SOA world, services are deployed within an enterprise, behind firewalls, by enterprise developers and deployers. End-users typically do not (and can not) deploy services. However, in the grid community deployment needs to be supported across firewalls and enterprise/organizational boundaries (i.e. inter-enterprise), for different types of deployers, some of whom are essentially end-users. For the deployment scenario, we assume: a set of Grid resources (possibly heterogeneous); shared by a number of VOs, but with at least one centralised index service for each VO listing the resources available to the users of that VO; end-user deployment; portable service/application code; and, manual deployment of 1st order services by Systems or Grid Administrators on each site.

3.1 Grid Service Deployment (1st order)

The scenario for grid service deployment is as follows: Configure service specific security; validate deployment; discover hosts available; select hosts to deploy to; deploy service to selected hosts; register the services in an index service**;** enable or disable the services. Related tasks include: Test to ensure that services are registered, discoverable and callable by specified users; un-deploy service; redeploy service; trace progress of deployment**;** debug deployment failure.

Security configuration. Given a GAR file, the deployer unpacks it, configures security for the target host, and then repacks ready to deploy. We assume that authentication is specified at development time in a custom service security configuration file. This allows the developer to specify the minimum level (and other permissible levels) of security for each service and method, but not *who* is authorized to use them. Authorization is specified in service specific gridmap files. A gridmap is an Access Control List that specifies which users have access to a service. It has a list of distinguished names and maps each name to a user account. The requirement to map user certificates to unique local accounts in gridmap files reduces the scalability of the deployment process. This is the default approach, but a number of alternatives are possible. Using role based security would reduce the complexity of managing gridmap files, but authentication is still a problem, requiring user certificates, proxy cer-

tificates, and certificate subjects. One simplification is to allow anonymous users (users who share the same credentials) so that nodes only need 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. It is possible that a common account could be used in gridmap files (every user would still have a unique certificate). Even though there would then be no privacy or isolation between users mapped to the same account (the common account functioning more as a "role") this may be a reasonable compromise.

Validation and Testing. Ideally the GAR file, deployment descriptors, and security configuration, could be validated before deployment, but this is not supported. Errors may be discovered during deployment, or worse, at run-time, which impinges on scalability, availability and reliability of deployment and use.

Host discovery and selection. We assume that services are portable, and that they will be deployed to all resources in a VO without reference to the base capacity, current resources, or load on each machine. This is a reasonable assumption since newly developed Grid Services are more likely to be portable compared with legacy applications deployed using MMJFS.

Deployment. For the experiment deployment was initially performed locally on each machine with the GT3 deployment scripts. The mechanism for manual deployment is to make the GAR file available to the deployer on each site and then wait for them to deploy it, restart the container, and register the service in the central VO index service. Restarting the container is problematic if services are in use unless they support persistence across container restarts. So-called "hot" deployment would be an advantage; otherwise a workaround is to have separate containers (or multiple Web Application Contexts using Tomcat) for each service, user, or VO. To undeploy/redeploy a service, the container is stopped, the service un-deployed, a new version of the service deployed if required, and then the container is started again. The MDS3 entry for the service must be updated or removed.

Service registration. MDS3 is designed to support a meta-data oriented registry service, various topologies (e.g. hierarchical aggregation), and soft-lifecycle management/update of service instance state changes. This makes it more than a simple UDDI registry service. It takes multiple steps to register a service in one container into a remote index service. This requires manual local server-side editing of configuration files, and information entered during configuration is not checked until execution time. In a typical SOA the directory service contains information about the service location, along with service description (WSDL). Our experiments did not require dynamic discovery of service descriptions since client-side code (including stubs) was developed at the same time as services. However, this capability is critical for scalable, flexible and robust SOAs, and we are obliged to assume that MDS3 supports registration and discovery of GWSDL service descriptions.

Non-functional deployment attributes. Using a manual deployment process the performance (time to deploy to a node), scalability (how many nodes can be deployed to with increasing nodes, services, and users), reliability, repeatability, traceability, and debuggability are all extremely poor. The security of deployment is only moderate since the process is manual and error-prone. It assumes secure transfer of GAR files to the deployers, that they are not tampered with by the deployers, and that only services from permitted developers are deployed. Validating the security configura-

tion of deployed services remotely is non-trivial. Scalability of security configuration maintenance is an issue, requiring authorized users to be added/removed from service specific security configuration files. This currently entails editing of grid-map files and then redeployment and restarting of the container. More seriously, in the absence of automated/remote deployment there is no formal security model for inter-organizational/VO deployment.

In the absence of any other resource management mechanism, deploying a service on a machine and giving users permission to use it gives them the "right" to consume resources on that machine simply by invoking it. The default service resourcing model is shared, not exclusive, but with no guaranteed QoS unless the hosting environment can provide it at the time of use taking into account both base-level capability and actual load. Extra resource scheduling or load-balancing mechanisms are required to ensure QoS, fair sharing of resources, and to prevent resource saturation.

3.2 2nd Order Application Deployment

The 2nd order application scenario is different, as using MMJFS an executable is deployed (or "staged") immediately prior to use as part of the same invocation of MMJFS by the same user. The steps are as follows: Prepare RLS-2 file based on application requirements; discover and select MMJFS services; call selected MMJFS with RSL-2 file; wait for success of staging, notification of job submission, and eventual termination.

MMJFS. MMJFS/GRAM is a basic job submission service (without scheduling) that takes an RSL-2 XML file and a user certificate as input, and submits the job to the underlying queue with the proper invocation syntax, running as the user account mapped to the certificate. MMJFS supports staging of executables using a GASS server which runs on the GRAM client and negotiates data transfers with the remote MJS service [12]. MMJFS services are assumed to be registered in a VO index service to be discovered at deployment time. We assume that deployment occurs to all of the resources available to a VO simplifying the problem of matching application requirements to resources (for example, platform and concurrency). Otherwise, the resource management system in each organization is responsible for discovering and allocating appropriate resources, although how this is coordinated globally across organizations is unclear. The distinction between Deployment and Use phases is somewhat artificial as file staging (i.e. deployment) is just one of the operations performed by MMJFS once invoked. The MMJFS Start Operation steps are as follows: client credential delegated to MJS instance, file staging performed, submit job to scheduler. There is a strong assumption that an application is deployed and then used immediately by the same user, although there may be a substantial delay before the job is executed if using a job scheduler. This limits the options for deploying/staging an executable in advance, splitting deployer and user roles, and may impact scalability, performance and flexibility [11].

Security. One distinction between the 1st and 2nd order approaches is security related to deployment – both configuration of security during deployment for subsequent use and security of deployment. The 1st order approach enables services and methods deployed in a container to have different security settings. Due to the lack of

an automated/remote deployment mechanism there is no explicit security model for deployment. The 2nd order approach imposes one security model on deployment and execution, due to the use of one mechanism for both tasks – i.e. the security configuration of MMJFS. Therefore only one set of security policies can be applied, to both the deployment and execution, of all jobs in a container. If a user has permission to use MMJFS in a given container, then they can deploy and use any application in that container. However, finer grained security may be provided by a resource manager and the use of virtual containers as sand-boxes would reduce security problems.

Data transfer, Index Services, and Tracing. GridFTP and a GASS server must be installed and working on servers and client machines. It is unclear if there is any explicit un-deployment capability and if/when or how files are cleaned up/deleted. MMJFS is already registered in the index service, but MJS instances (returned from the MMJFS Create Service operation) can also be registered to enable management of individual jobs (allowing for long-running batch jobs). There is some support for tracing the progress of MMJFS events and exceptions since MMJFS was designed to manage job execution. However, only minimal information is available remotely.

3.3 Analysis

Remote deployment of "applications" is straightforward with the 2nd order approach, although more infrastructure must be installed (MMJFS, security, resource manager, data transfer services). There is support for resource matching and some support for deployment tracing/debugging. There is no capability for "application" registration, explicit deployment packaging, or validation of the deployment prior to use. There is an explicit security model for deployment, which is just the MMJFS security settings, and therefore identical for deployment and job submission for the whole container. Deployment and Use are therefore indivisible, both temporally, for identity/authentication, and for authorization. There is some support for tracing/debugging, but it is impossible to test MMJFS deployment without security being enabled. Deployment at least guarantees job submission and therefore (eventually) resourcing.

There is no in-built support for remote deployment of Grid Services and therefore no formal model of deployment security, no support for resource matching (although portability of services can reasonably be assumed), and very poor non-functional deployment characteristics. There is explicit Grid Service deployment packaging (GAR file) and it would be possible in principle to validate at least parts of the deployment prior to, or during, deployment. Service registration is well supported and we assume that it is possible to register GWDSL service descriptions. The scalability of the default 1st order service security model is poor, requiring configuration for each site to map local accounts to user certificates. However, testing service deployment is feasible prior to security being enabled and a simpler more scalable security model may be possible. Deployment allows for immediate invocation but does not guarantee QoS.

4 Remote Deployment with SmartFrog

Due to the lack of support for automated remote deployment of Grid infrastructure and services across organizations in the Globus middleware stack we trialled SmartFrog (a 3rd party component deployment technology [14]) for Grid deployment and conducted five experiments as follows: intranet deployment; internet deployment; secure deployment; deployment of secured infrastructure; and deployment of services.

Deployment on an Intranet: Within the laboratory. A project at UCL [15] investigated the use of SmartFrog to deploy GT3.2 on an intranet in a laboratory setting. This involved: configuring SmartFrog to remotely install and start the core GT3 package and Tomcat container; deploy sample grid services across multiple machines in the laboratory; and the development of a management console to control the process. The solution worked well in the laboratory but relied on the freedom to install and run a new installation of GT3 and 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), with an identical configuration for each installation. A GUI management console was provided for selecting target machines (based on available resources) and installing, configuring, starting and stopping the infrastructure or services. The progress of the installation along with any exceptions could be monitored and a partial (services) or complete (infrastructure) uninstall performed. The deployment process was scalable for increasing numbers of machines and was portable across different versions of Linux/UNIX.

Deployment on the Internet: Across sites and firewalls. Given the success of the intranet experiment we moved out of the laboratory setting and applied the experimental SmartFrog infrastructure across the internet to the OGSA test-bed sites. However, the new environment introduced a number of difficulties. We were unable to get the collection of components developed in the laboratory (a version of SmartFrog, Grid specific deployment files, and GUI management console) working together correctly across sites, although deployment was demonstrated across an unsecured port in a limited test situation using the default unmodified SmartFrog package and examples. Because of differences in site security policies and the perception that SmartFrog is a perfect virus propagation mechanism it was impossible to open a new SmartFrog daemon listener port across all the test-bed sites. In theory RMI over HTTP (tunnelling [16]) could be used over the already open grid container port, but secure deployment was still a precondition.

Secure deployment. SmartFrog and Globus use different security models and certificates. In order to deploy infrastructure securely with SmartFrog an independent (and therefore redundant) security infrastructure, process, and certificates is 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 well designed for the deployment domain. Issues related to SmartFrog security configuration, use, and debugging prevented us from getting it working correctly across sites, illustrating the difficulties of debugging security infrastructure, and secured infrastructure. Security and debugging are mutually exclusive.

Deployment of Secured Infrastructure. The next challenge was to use SmartFrog to install, configure and run a *secure* GT3 installation and container. The first problem is the requirement for the deployment infrastructure to have access to host certificates, user certificates, and local accounts, and to prepare customised deployment configurations (e.g. the gridmap files) for each site. 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 [17]. However, there are significant issues to do with trust, security and auditing if the binding between users and accounts is weakened. A role-based security mechanism is an alternative [18]. A fundamental obstacle is if the SmartFrog daemon needs to run as a privileged user such as "root", as is the case for configuration of the standard GT3 production security environment. The "–noroot" option is an alternative for configuring GT3 security without root permission, but it is unlikely to be appropriate for production environments [19]. One workaround for the security installation problem is for the first installation and security configuration to be done locally and manually, enabling subsequent updates or (re-)installations for different user communities to proceed automatically/remotely by reusing the first security configuration. As long as there is one correctly installed and secured version, other versions (of at least the GT3 core package) can be installed by non-root users.

Deploying Services. The final goal was deploying *services* to an already deployed infrastructure. In the laboratory SmartFrog demonstrated the ability to deploy services to a container and then start the container. However, because of the lack of "hot" deployment in GT3 stopping and restarting a container that is in use is unlikely to be acceptable. "Hot" deployment and/or running multiple real (or virtual) containers (one per user or VO) are possible tactics. However, another problem is deploying *secured* services since these may require both service and site specific configuration.

5 Related Work

This section reviews related work in deployment and security. Because of the functionality available at deployment time and the complexity of deployment, deployment is an explicit role in the EJB/J2EE specification [20] and is supported in J2EE products. Some products go further than the specification and provide remote deployment, automatic updating of client-side code from a server, and one-step deployment of components to a cluster. Java Web Start and the underlying Java Network Launch Protocol (JNLP) provide a simple way of end-users installing and running new (client-side) Java programs over the Web [21]. Operating System patch management systems such as Microsoft's Windows Update could be applied to middleware [22].

In the Java community there is a view that J2EE is too heavy-weight and POJOs (Plain Old Java Objects) are enough. With the support of light-weight containers such as Spring/Hibernate POJOs can be deployed with close to zero effort, as deployment dependencies are resolved by containers using reflection [23]. Inversion of Control (IoC) and Aspect Oriented Programming (AOP) approaches to component portability could be applied to Grid deployment [24, 25].

Problems with deployment in Globus have been documented [26], as have Grid deployment Use Cases which complement our deployment scenarios [27, 28]. Other approaches and tools for Grid deployment include GITS [29], distributed Ant [30], the IBM autonomic deployment framework [31], deployment planning [32], PACMAN [33], GPT [34], and Virtual Machines [35]. None of these deal adequately with the deployment of services.

Work that specifically targets Grid service deployment includes model based deployment [35], dynamic deployment [37], QoS-aware deployment [38], dynamic service architecture [39], hot service deployment [39], grid service GUI [41], remote deployment interface [42], and two projects using SmartFrog [43, 44]. Related work on web services deployment includes remote deployment in Tomcat [44], Axis [46], and P2P web services deployment [47]. However, despite acknowledging the importance of the problem and providing a variety of solutions, we do not believe that any single existing approach adequately deals with all aspects of secure deployment of secured Grid infrastructure and services across firewalls.

An increasing number of projects are working on solutions to security issues and better tools and procedures are likely [11, 18, 47-50]. However, it is critical to ensure that these work seamlessly with services. CAS [50] does not work with grid services.

6 Conclusions

Remote grid deployment infrastructure needs to: support deployment of infrastructure and services across organizational boundaries and firewalls; be secure; be able to deploy secure infrastructure; be manageable (deployment state, progress and errors monitored, and debugged and fixed); support configuration and version management, recovery and audit trails; be reliable and repeatable; maintain consistent versions and sets of components and services for a VO across heterogeneous resources; support multiple scenarios (e.g. un-deployment), roles, and role/trust delegation; be scalable (with increasing users, nodes, and services); be usable (easy to install, use and administer, portable, GUI tool support).

A fundamental problem is how to bootstrap the installation process. Which comes first: The deployment infrastructure or the grid infrastructure, the deployment security or the infrastructure security? On the face of it, the easiest solution is to start with a light-weight, portable, easy to install and secure, deployment infrastructure which is then used to bootstrap the installation of the Grid infrastructure, security and services. This is the approach we took with SmartFrog and which was demonstrated to work in the laboratory albeit with a number of simplifying assumptions. However, out of the laboratory, installing, configuring, securing, and debugging extra and redundant infrastructure for deployment presents the same types of problems as does the installation of grid middleware itself. The duplication of the security infrastructure and extra issues of trust by Systems Administrators, and use of the deployment infrastructure to secure grid infrastructure are significant barriers to this approach.

An alternative approach is to first remove the requirement for a redundant deployment-specific security infrastructure by using a lightweight security mechanism as the core of both the deployment and grid infrastructure. This allows the security mecha-

nism to be set up once correctly and then used as the basis of deployment and infrastructure security. We believe this is feasible as the security requirements for 1st order service security and deployment are simpler, or at least different, to what the GSI model is designed for. Ideally the security model would be composable (or extensible) so that its capabilities could evolve [53]. We have observed that the core GT3 package is relatively lightweight and portable compared with the other packages. It is therefore possible to remove the requirement for a redundant deployment infrastructure by including basic remote deployment capabilities in the core GT3 package. Assuming initial manual deployment of the core grid infrastructure, including basic deployment and security, the grid infrastructure itself can then be relied upon to support subsequent remote automated infrastructure re-installation/updates and service deployment. That is, deployment is a first-class citizen and adding it as an afterthought, or as an extra redundant infrastructure is best avoided. It needs to be built into the middleware stack. It would also be feasible to expose the middleware's remote deployment capability as a "service deployment service" in the container (using SOAP attachments to transfer GAR files). Finally, we suggest that the problem of debugging and rectifying run-time failures can be (partially) solved by making critical deployment context information available at run-time, along with the ability to redo some of the deployment steps. We call this approach "Deployment-aware debugging" which will be addressed in another paper.

Building a loosely coupled distributed grid system across organizational boundaries using OGSA is non-trivial and different from building a system over a LAN. This paper demonstrates that there is a need for better understanding of, and support for, cross-cutting non-functional inter-organizational roles such as deployment. There is a lot more work to do before we realize the vision of the Grid.

References

1. The UK-OGSA Evaluation Project. http://sse.cs.ucl.ac.uk/UK-OGSA/
2. Foster, I., Kishimoto, H., Savva, A. (eds.): The Open Grid Services Architecture, Version 1.0 (2005). http://www.gridforum.org/documents/GFD.30.pdf
3. Globus Toolkit 3.0. http://www-unix.globus.org/toolkit/3.0/ogsa/docs/
4. Brebner, P. (ed.): UK-OGSA Evaluation Project Report 1.0: Evaluation of Globus Toolkit 3.2 (GT3.2) Installation (2004). http://sse.cs.ucl.ac.uk/UK-OGSA/Report1.pdf
5. Brebner, P. (ed.): UK-OGSA Evaluation Project Report 2.0: Evaluating OGSA Across Organizational Boundaries (2005). http://sse.cs.ucl.ac.uk/UK-OGSA/Report2.pdf
6. Brebner, P., Two Ways to Grid: A Service-centric vs. Resource-centric evaluation of the Open Grid Services Architecture (OGSA), CSIRO Technical Report (2005). http://www.ict.csiro.au/staff/Paul.Brebner/TwoWaysToGrid.htm
7. S. Tuecke, et. al.: Open Grid Services Infrastructure (OGSI) Version 1.0. Global Grid Forum Draft Recommendation (2003)
8. An Interview with Argonne's Steve Tuecke. IBM developerWorks (2003). http://www-106.ibm.com/developerworks/java/library/j-tuecke.html?dwzone=java
9. UK e-Science Certification Authority. http://www.grid-support.ac.uk/ca/
10. Girard, J.: Staging Files for Grid Jobs using Globus GASS Server. IBM developerWorks (2003). http://www-106.ibm.com/developerworks/grid/library/gr-cglobus3/

11. Workspace Management Service. http://www-unix.mcs.anl.gov/workspace/tech_preview_2/docs/index.html
12. Elwasif., W., Plank, J., Wolski, R.: Data Staging Effects in Wide Area Task Farming Applications. IEEE International Symposium on Cluster Computing and the Grid. Brisbane, Australia (2001)
13. Yahyapour, R.: Grid Resource Management and Scheduling. Europar 2004 Tutorial. http://www.di.unipi.it/europar04/Tutorial3/Europar_Tutorial_GRMS_Yahyapour.ppt
14. Goldsack, P., Guijarro, J., Lain, A., Mecheneau, G., Murray, P., Toft, P.: SmartFrog: Configuration and Automatic Ignition of Distributed Applications. HP (2003). http://www.hpl.hp.com/research/smartfrog/papers/SmartFrog_Overview_HPOVA03.May.pdf
15. Kong, D., Novov, V., Tsalikis, D., Koukoulas, S., Karampaxoglou, T.: Deployment in Computational Distributed Grids. Main Report. UCL MSc Data Communications, Networks and Distributed Systems Project (2004).
16. jGuru: Remote Method Invocation. Sun Developer Network. (2000). http://java.sun.com/developer/onlineTraining/rmi/RMI.html
17. Globus 3.2.1. Job Submission Errors. Globus-discuss (2004). http://www-unix.globus.org/mail_archive/discuss/2004/10/msg00276.html
18. The PERMIS project. http://www.permis.org/en/index.html
19. GT3.2 Installation Guide. http://www-unix.globus.org/toolkit/docs/3.2/installation/install_installing.html#rootNonroot
20. Enterprise JavaBeans Specification, Version 2.1. Sun Microsystems, http://java.sun.com/products/ejb/docs.html
21. JNLP. http://java.sun.com/products/javawebstart/faq.html, http://java.sun.com/developer/technicalArticles/Programming/jnlp/
22. Dadzie, J.: Understanding Software Patching. ACM QUEUE. March (2005)
23. Matthew, S.: Examining the Validity of Inversion of Control. The Server Side. (2005). http://stage.theserverside.com/articles/article.tss?l=IOCandEJB
24. Fowler, M.: Inversion of Control Containers and the Dependency Injection Pattern. (2004). http://www.martinfowler.com/articles/injection.html
25. Spille, M.: Inversion of Control Containers. (2004). http://www.pyrasun.com/mike/mt/archives/2004/11/06/15.46.14/index.html
26. C. Mattmann, S. Malek, N. Beckman, M. Mikic-Rakic, N. Medvidovic and D. Crichton. GLIDE: A *G*rid-based, *L*ightweight, *I*nfrastructure for *D*ata-intensive *E*nvironments. European Grid Conference (EGC2005), pp. 68-77. Amsterdam, February (2005)
27. Lamanna, M., Rocha, R.: Grid Deployment Use Cases. LHC CERN (2004). http://lcg.web.cern.ch/LCG/peb/GTA/GTA-ES/es-008.doc
28. Foster, I., Gannon, D., Kishimoto, H., Von Reich, J. (eds.): OGSA Deployment Use Cases. Global Grid Forum (2004). http://www.ggf.org/documents/GWD-I-E/GFD-I.029v2.pdf
29. Fenglian X., Eres, M., Baker, D., Cox, S.: GITS, Grid Integration Test Script. IEEE International Conference on Services Computing (2004) 281 - 287
30. Goscinski, W., Abramson, D.: Distributed Ant: A System to Support Application Deployment in the Grid. IEEE/ACM International Workshop on Grid Computing (2004) 436-443.
31. Small, L.: The IBM autonomic deployment framework. http://www-128.ibm.com/developerworks/autonomic/library/ac-abc2/
32. Lacour, S., Perez, C., Priol, T.: Deploying CORBA Components on a Computational Grid: General Principles and Early Experiments Using the Globus Toolkit. In: Emmerich, W. Wolf, L. (eds.): Proceedings of the 2nd International Working Conference on Component Deployment (CD 2004). Number 3083 LNCS. Edinburgh, Scotland, UK. Springer-Verlag (2004) 35-49

33. PACMAN: http://physics.bu.edu/~youssef/pacman/
34. Grid Packaging Tools (GPT): http://www.ncsa.uiuc.edu/Divisions/ACES/GPT/
35. Childs, S., Coghlan, B., O'Callaghan, D., Quigley, G., Walsh, J.: Deployment of Grid Gateways using Virtual Machines. Proceedings EGC'05. Amsterdam (2005). https://www.cs.tcd.ie/coghlan/pubs/egc-vm-deployment.pdf
36. Huang, G., Wang, M., Ma, L., Lan L., Liu, T.: Towards architecture model based deployment for dynamic grid services. IEEE International Conference on E-Commerce Technology for Dynamic E-Business (2004) 14 – 21
37. Ting, A., Caixia, W., Yong, X.: Dynamic Grid Service Deployment (2004). http://www.comp.nus.edu.sg/~wangxb/SMA5505-2004/xieyong-report1.pdf
38. Musunoori, S., Eliassen, F., Staehli, R.: QoS-aware component architecture support for grid. WET ICE 2004. 13th IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (2004) 277 - 282
39. Weissman, J.: Enabling communities of collaborating users and services on the Grid. http://www.dtc.umn.edu/resources/weiss.ppt#1
40. Friese, T., Smith, M., Freisleben, B.: Hot service deployment in an ad hoc grid environment. ICSOC (2004) 75-83
41. Wood, M., Ferner, C., Brown, J.: Towards a GUI for Grid Services. Proceedings of the IEEE Southeastern Conference. Greensboro NC (2004) 316-324 http://people.uncw.edu/cferner/papers/IEEESECON2004_047.pdf
42. Wu, Y.: CGSP 1.0 (China Grid Support Platform). Asia Summit Grid (2005). http://www.gridforumkorea.org/asiagridsummit2005/data/WuYongWei.pdf
43. Talwar, V., Milojicic, D., Wu, O., Pu, C., Yan, W., Jung, G.: Approaches for Service Deployment. IEEE Internet Computing Vol. 9 No. 2 March/April (2005).
44. Anderson, P., Smith., E.: OGSAConfig. http://groups.inf.ed.ac.uk/ogsaconfig/
45. Tomcat Manager. http://jakarta.apache.org/tomcat/tomcat-4.0-doc/manager-howto.html
46. CypressLogic ObjectView Axis Deployment Product. http://www.cypresslogic.com/home.html
47. Harrison, A., Taylor, I.: WSPeer - An Interface to Web Service Hosting and Invocation. 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2005)
48. Beckles, B.: Removing digital certificates from the end-user's experience of grid environments. UK eScience All Hands Meeting (2004)
49. Virtual Organizations Membership Service (VOMS). http://edg-wp2.web.cern.ch/edg-wp2/security/voms/
50. Pearlman, L., Welch, V., Foster, I., Kesselman, C., Tuecke, S.: A Community Authorization Service for Group Collaboration. 3rd International Workshop on Policies for Distributed Systems and Networks. Monterey, California. IEEE (2002).
51. Emmerich, W., Butchart, B., Chen, L., Wasserman, B., Price, S.: Grid Service Orchestration using the Business Process Execution Language (BPEL). Submitted to Journal of Grid Computing. (2005)
52. Lamport, L.: http://research.microsoft.com/users/lamport/pubs/distributed-system.txt
53. Llewellyn-Jones, D., Merabti, M., Shi, Q., Askwith, B.: Secure Component Composition for Personal Ubiquitous Computing. ProgNet Workshop (2003). http://www.cms.livjm.ac.uk/pucsec/dnload/pucsec02.pdf
54. Brebner, P.: Grid Middleware: Principles, Practice and Potential. UCL Computer Science Department Seminar (2004). http://sse.cs.ucl.ac.uk/UK-OGSA/GridMiddlwarePPP.ppt