Two Ways to Grid: The contribution of Open Grid Services Architecture (OGSA) mechanisms to Service-centric and Resource-centric lifecycles[1]

Paul Brebner[2], Wolfgang Emmerich[3]

[2] CSIRO ICT Centre, Australian National University, Canberra, ACT 2601, Australia
Paul.Brebner@csiro.au
[3] Department of Computer Science, University College London, London WC1E 6BT, United Kingdom
W.Emmerich@cs.ucl.ac.uk

Key words: Open Grid Services Architecture (OGSA) evaluation, Service Oriented Architecture (SOA) lifecycles

Abstract. Service Oriented Architectures (SOAs) support service lifecycle tasks, including Development, Deployment, Discovery and Use. We observe that there are two disparate ways to use Grid SOAs such as the Open Grid Services Architecture (OGSA) as exemplified in the Globus Toolkit (GT3/4). One is a traditional enterprise SOA use where end-user services are developed, deployed and resourced behind firewalls, for use by external consumers: a Service-centric (or "1st order") approach. The other supports end-user development, deployment, and resourcing of applications across organizations via the use of execution and resource management services: a Resource-centric (or "2nd order") approach. We analyze and compare the two approaches using a combination of empirical experiments and an architectural evaluation methodology (scenario, mechanism, and quality attributes) to reveal common and distinct strengths and weaknesses. The impact of potential improvements (which are likely to be manifested by GT4) is estimated, and opportunities for alternative architectures and technologies explored. We conclude by investigating if the two approaches can be converged or combined, and if they are compatible on shared resources.

1 Introduction

A Service Oriented Architecture (SOA) supports the service lifecycle tasks of development, deployment, hosting and registration, and discovery and invocation [1]. In a cross-organizational Grid SOA, installation of a common (or at least, interoperable) infrastructure across the participating Grid nodes is a critical prerequisite task. In an

[1] A longer version of a paper submitted to ICSOC 2005.

Enterprise SOA, there are two distinct roles associated with these tasks: Provider and Consumer. The Provider develops, deploys, hosts, registers and manages services behind a firewall, for internal or external consumption. The Consumer discovers and uses services, and may be internal or external to the organization. The provider role is therefore exclusively intra-organisational, while the consumer role is intra- and inter-organizational.

However, with Grids the story is different. End-user scientists typically want to develop their own applications, find resources to run them on, deploy and execute them on those resources, and manage their own applications. The focus is on end-user development, deployment and use, and therefore overlapping provider and consumer roles which cross organisational boundaries and firewalls.

The Open Grid Service Architecture (OGSA [2]) is designed to support Grid SOA lifecycles, and specifies high-level services, motivated by both functional and non-functional Grid requirements, including:

- Infrastructure services
- Execution Management services
- Data services
- Resource Management services
- Security services
- Self-Management services
- Information services.

These OGSA Services will be built on core infrastructures. This paper is based on a cross-organizational experimental evaluation [3] and a subsequent analysis [4] of one exemplar OGSA infrastructure, the Globus Toolkit (GT3.2), based on the Open Grid Services Infrastructure (OGSI [5]).

OGSA and GT3 are based on Web Services standards, but deviate or build upon them in a number of important ways. GT3 uses GWSDL, an extension of WSDL 1.1, and Grid Service Factories and Grid Service Handles (GSH) to support explicit management of stateful service lifecycles. Grid Security Infrastructure (GSI) is the security mechanism used by GT3, which supports PGP signing/encryption, client and host certificates, proxy certificates, and delegation. The Globus Monitoring and Discovery System (MDS3) is an enhanced index service which supports management by exposing the state of service instances with Service Data Elements (SDEs). The Master Managed Job Factory Service (MMJFS) deploys, resources, and executes arbitrary applications, and can be used in conjunction with 3rd party resource schedulers.

GT3 supports end-user/consumer deployment scenarios using the mechanisms of Grid Services, GSI, MMJFS, Resource Specification Language (RSL-2), data transfer services (E.g. GridFTP, the Grid File Transfer Protocol, and GASS, Global Access to Secondary Storage protocol) and index services (MDS3). For authorized users these allow:

- job submission services (MMJFS) to be discovered
- arbitrary code and data to be transferred onto the target machine
- resource requirements to be specified
- resources to be scheduled and allocated

- jobs to be submitted, run and monitored
- notification of completion
- across organisational boundaries and firewalls.

However, there is no support for cross-organisational end-user deployment of "1st order" grid *services*. This means that science code cannot easily be deployed by an end-user as a Grid service according to the following scenario:

- implemented as a Grid Service
- described (in GWSDL)
- packaged (in a Grid Archive, or "GAR" file)
- deployed across organisations to all the containers available to a Virtual Organisation (VO)
- registered in a VO index service
- discoverable and callable by all the authorized users in a VO.

Support for deployment of Grid Services across firewalls is lacking in GT3 because of fundamental differences in perception about how Grid Services, applications, and infrastructure will be used. Traditionally, scientific programs are large monolithic applications written in legacy languages, which are difficult to port to a 1st order Grid Service model. GT3 is designed as a resource-centric infrastructure to allow legacy applications to portably utilize heterogeneous Grid resources. Moreover, there is no default load balancer or resource manager (either local or global) for 1st order grid services. This highlights the difference between two uses of Grid infrastructures: On-demand just-in-time grid-enabling of Web Services (i.e. resourcing *of* services, or Service-centric); and Web Services enabling of Grids (i.e. resourcing *by* services, or Resource-centric). We will subsequently refer to these as "1st order" and "2nd order" uses of OGSA. Note that these are our terms, and were chosen to characterize the Enterprise approach of exposing business functionality as "1st order" services, versus the Grid approach of using services to resource applications (so called "2nd order" as the application is not exposed directly as a service, only being available indirectly via generic services). Figure 1 illustrates the two approaches. Triangles represent external service interfaces, the large circles are containers, which, for the 1st order approach hosts the implementation (small circles) of utility services (e.g. Index) and science services, and for 2nd order, hosts the utility services (e.g. Data, Index and Execution services) which are used to resource, and control externally hosted science applications.

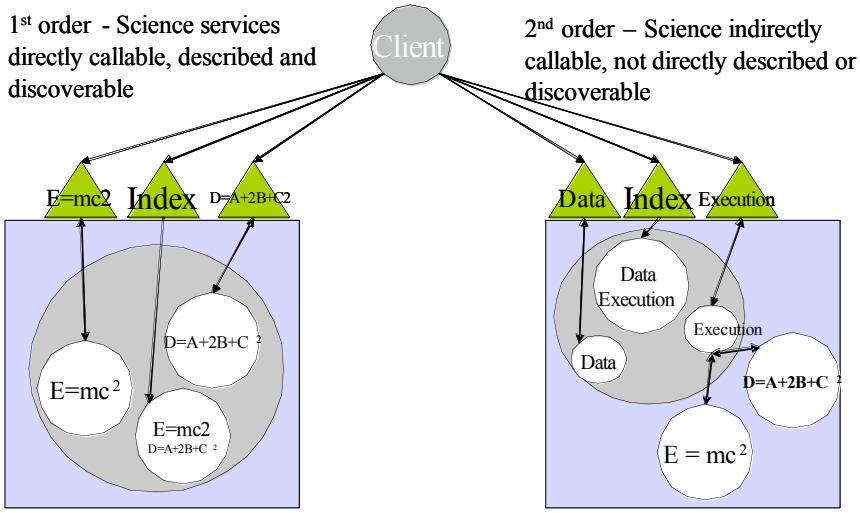

Figure 1 1st vs. 2nd order Uses of OGSA

We investigate two different ways of implementing the same benchmark application using OGSA [6]. The first way exposes science as "first-order" grid services, and the second way treats science as legacy code to be executed by MMJFS. We use a scenario, mechanism and quality attribute methodology to evaluate and compare the two approaches. The main qualities of interest are aligned with the primary SOA tasks of Installation, Development, Deployment and Use. These are defined by logical scenarios, and the main steps of the scenarios, and some non-functional attributes, are evaluated based on the contribution of the 1st and 2nd order grid mechanisms to primitive attributes from which they are composed. The value "1" is used for mechanisms that help/contribute, "-1" for those that hinder or have a side-effect, "0" for those that are indifferent (or, in a few cases, where there was substantial uncertainty about the effectiveness of a mechanism, typically due to lack of experimental verification), otherwise "N/A" for those that are irrelevant. The average contribution of the mechanisms is then used to determine a composite quality for each task. For details of the contribution of the primitive attributes see the Appendix and [4].

Early on in the evaluation Globus announced that development of GT3 would cease, and that it would be superceded by GT4, which would be based on the Web Services Resource Framework (WS-RF) rather than OGSI. The evaluation was therefore designed so that the results would be more generally applicable, applying at least to GT4. This was achieved by combining both empirical measurement, and an architectural analysis method (which emphasizes architectural features and technology mechanisms, rather than focusing on purely implementation artifacts), and by introducing a "What if" scenario to explore the impact of expected improvements. This evaluation contributes to an understanding of the differences between the two uses of Grid SOAs, and baselines the strengths and weaknesses of the various mechanisms of the pioneer OGSA implementation *as we experienced it for a 12 month snapshot at the end of its development cycle*. We expect that it will be useful to measure future

improvements against, and to predict the potential impact of changes to usage styles and underlying technologies.

We assume a "null" working hypothesis. That is, that there is no significant difference between the two approaches. This is tested by comparing the two approaches for each task (sections 2-5). We summarise the strengths and weaknesses of each approach and investigate the potential impact of improvements (section 6), review related work at the boundaries of component, web and grid services (section 7), and explore future scenarios including convergence, combination, and compatibility (section 8).

2 Installation

2.1 Scenario and Attributes

As part of the UK OGSA evaluation project, GT3 infrastructure was installed across four test-bed sites [3] on heterogeneous platforms in the presence of different site-specific security, access and firewall policies. GT3 has a complex package structure, and it was initially unclear what packages were dependent on each other or were required to achieve different sets of functionalities. Just the "core package" (containing pure Java container related services only) and a Tomcat container is required for 1st order services, but in order to provide security, at least part of the "All Services" package (containing the entire middleware stack) was installed on some sites. The common components required for both approaches are: Core package, SOAP server, container, security, index service. Extra components required for the 1st order approach are an optional load-balancer and SOAP attachments, and for the 2nd order approach the MMJFS service, optional resource scheduler and data transfer services. We did not install either a load-balancer for services or a resource scheduler for MMJFS as these are not part of the default GT3 distribution. The installation task scenario steps are as follows: Server-side install and configure; Configure security; client-side installation.

For each approach, the contribution of each component to the following non-functional attributes for each step was evaluated:

- Package (how easy is it to obtain the installation components)
- Independent (is other supporting infrastructure needed)
- Portable (can it be installed on heterogeneous platforms)
- Build (how easy is it to build the first time)
- Repeatable and reliable build (can the build be repeated easily)
- Configure (how easy is it to configure deployment information)
- Test (how easy is it to test the installation)
- Scalable (how easy is it to repeat installation for multiple nodes)
- Manage (how easy is it to view, manage, and change state of installation)
- Remote manage (can management be performed remotely).

2.2 Evaluation Summary

Installation of infrastructure to support the 1st order approach is easier than for 2nd order approach in theory, as there is less of it, and it does not need support of legacy components (and so is more portable). Portability and packaging of infrastructure is an issue in GT3. There is a need for well supported binaries or portable code (i.e. all Java), better integration and packaging of components, and a portable client-side package. 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, as only a container and security are essential, whereas the 2nd order approach relies on legacy components at present, which are less portable and require more effort to install, configure and test before use. A serious problem for security scalability is creating and mapping local accounts to user certificates for each node. For further details see the Appendix of [27]. Security infrastructure processes and management, including certificates and accounts, need improvement to be more useable, scalable and capable of automation. Better support for role-based security would be a step forwards. Improvements in processes, tools and technologies are required to support remote automatic installation and configuration; monitoring of installation progress and state; visibility of components installed and versions, and security infrastructure; and debugging of installations; for either approach. Automation of secure remote installation of secured infrastructure is non-trivial, as it involves bootstrapping the installation, including security infrastructure. Security requirements and node-specific configuration are impediments to automating infrastructure installation.

2.3 Analysis

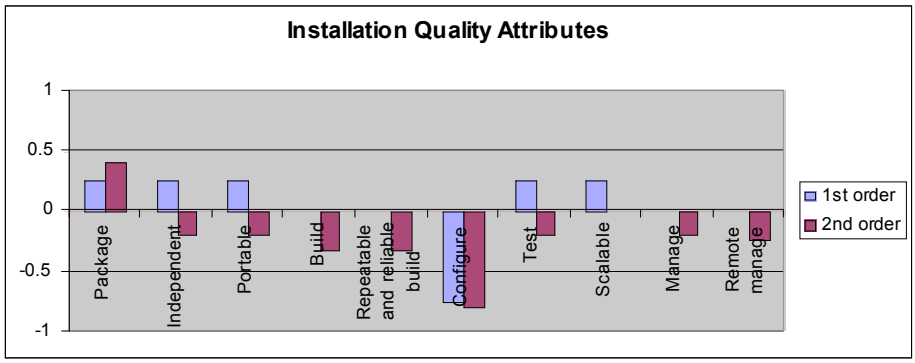

Figure 2 Installation Quality Attributes

Figure 2 shows the average contribution of the applicable components and steps to the installation quality attributes for 1st and 2nd order approaches. The 1st order components contribute to packaging, independence, portability, testability, and scalability, and only hinder configurability. The 2nd order components hinder independence, portability, buildability, configurability, testability, and manageability. 2nd order packaging is slightly better, mainly due to the contribution of the "All Services" package (which packages everything!). Comparing the two approaches, 1st order is better for independence, portability, buildability, testability, scalability, and manageability. None of the components contribute in a significant cumulative way to ease of installation however, and configurability is a major impediment.

Table 1 Installation evaluation

Attributes	1st order	2nd order
Core package	0.625	0.625
Security	-0.5	-0.5
MDS3	0.5	0.5
Client side installation	-0.285	-0.555
All Services package	N/A	-0.8
Overall	**0.08**	**-0.15**

Table 1 shows the contributions (averaged across all attributes) of the major components, steps, or packages. The common components (core, security, and MDS3) obviously have identical contributions. 1st order client side installation is marginally easier than 2nd order. However, the most significant observation is that only the core package and MDS3 contribute significantly to ease of installation. Security, client side installation and "All Services" hinder, although there is potential for security and client side installation to be improved for the 1st order approach. Overall, neither approach contributes significantly to installability.

3 Development

3.1 Scenario and Attributes

Two critical issues in building and using grid infrastructures are Interoperability and Implementability. In general, interoperability is achieved by conformance to protocol and interface standards. GT3 addresses interoperability by utilising Web Services standards although these are peculiarly Grid specific. It tackles the problem of ease of development of services by extending the Web Service model with a rich component model. This includes a set of conventions for service naming and reference, common and extended interfaces and behaviours, and container-based run-time support to manage service lifecycles. A downside of run-time support is the requirement to package and deploy components into a container environment. The mechanisms used

for 1st order development are Grid Services, GWSDL, and GAR files, and for 2nd order the mechanisms are Application executables and RSL-2 XML files. The logical development task scenario steps are as follows: Implement service; Describe interfaces; Configure deployment information; Package; Document resource requirements. A related task is: Develop client side code. For the 2nd order approach the scenario is simpler: Implement (use existing code); Describe interfaces and resources required. The attributes of interest correspond to the development steps as follows:

- Implementability (taking into account the component model, run-time support, container management, component types, component safety, portability, and legacy/binary code use)
- Interface description (expressiveness, ease of use, standardization)
- Configuration using deployment descriptor (expressiveness, ease of use, standardization)
- Packaging (expressiveness, ease of use, standardization)
- Resource requirements (expressiveness, ease of use, standardization)
- Client development (packaging, security, independence, client types)
- Tool support (adequacy, independence, standardization).

3.2 Evaluation Summary

The prerequisites for 2nd order development are minimal and it is naturally suited to legacy applications as no implementation effort is required, binaries can be re-used directly, and configuration and packaging steps are omitted. However, there is still some effort to construct the RSL-2 file, and portability depends on the application. For a new, non-legacy application, there is limited potential for component-based development, as run-time support is unavailable. There is poor support for rich client side code such as work-flows.

For 1st order development Gobus jars and a development environment which supports packaging are required. The component model is rich but non-standard, and potentially un-safe to use. However, it is portable and provides some run-time support from containers. Producing GWSDL and deployment descriptors by hand is time-consuming and error-prone, and becomes increasingly impossible for legacy applications (due to not being designed as services, badly documented or understood, and have complex calling semantics). There is no capacity for resource specification for services and there is little tool support for complete lifecycle development. Client side code development requires Globus jars and programmatic security configuration, but can support a variety of client types including work-flows.

3.3 Analysis

Table 2 Development evaluation

Attributes	1st order	2nd order
Implement	0.428	0.333

Interface	0.333	-0.333
Configure	0.5	N/A
Package	0.333	N/A
Resource requirements	N/A	0.333
Client	0	-0.75
Tool support	-0.333	N/A
Overall	**0.21**	**-0.10**

Table 2 shows the contribution of the 1st and 2nd order mechanisms to the development scenario steps, including an extra activity for "tool support". Implementation is roughly equivalent, because 2nd order requires no effort for legacy code implementation, but portability depends on the code, and 1st order provides a richer portable component model, with run-time support. Both have an interface definition language (GWSDL and RSL-2), but neither is standard. 2nd order has a resource documentation model (RSL-2), but this is largely irrelevant for 1st order as we assume OGSA components are portable (although there may be architectural and operational reasons why deployment to specific resources is required). 1st order clients can be richer (e.g. workflows), are better supported and require less client-side development (and run-time) infrastructure than 2nd order clients. 1st order configuration and packaging are well supported but note that because these steps are not required for the 2nd order approach they incur extra effort.

2nd order description hinders development because RSL-2 has no tool support, is non-standard, and there is no explicit relationship between it and the application it describes. The 1st order mechanisms for configuration, packaging and tool support have no counterpart for 2nd order development. The lack of tool support hinders development. 2nd order resource specification has no counterpart in 1st order, and contributes to development. 1st order client mechanisms do not contribute directly to development (with a value of "0"), but 2nd order client mechanisms hinder development.

The 1st order mechanisms used for implementation, description, configuration, and packaging all contribute to development. Client-side development mechanisms are neutral, but overall tool support hinders development. Only implementation and resource specification mechanisms contribute to 2nd order developability, while interface description and client-side mechanisms hinder development. Overall, the 1st order mechanisms weakly contribute to developability (0.21) compared with the 2nd order mechanisms (-0.1).

4 Deployment

4.1 Scenario and Attributes

In the enterprise SOA world deployment of new services is done within an enterprise, behind firewalls, by enterprise developers and deployers. Deployed components are exposed as services for intra- and inter- organisational interoperability. End-users

typically do not (and can not) develop or deploy code. However, in the grid community deployment must be supported across firewalls and organisational boundaries for end-users. For the deployment scenario we assume a set of heterogeneous Grid resources, 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 initiated deployment; portable service/application code; and, manual deployment of 1st order services by Systems or Grid Administrators on each site (given the lack of a remote GT3 service deployment mechanism). Mechanisms for 1st order deployment are manual local deployment, GAR file, container, and index service. Mechanisms for 2nd order deployment are "All Services" package, SOAP server, container, index service, MMJFS, security infrastructure, data transfer service/GASS server, application code and RSL-2 file.

The abstract deployment task scenario steps are host discovery and selection, deployment and enablement, and registration. The 1st order deployment steps are configure security settings, validate deployment, discover and select hosts, deploy services to selected hosts, register services in VO index service, and enable/disable services. The 2nd order approach to deployment with MMJFS is different as 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 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 and notification of job submission, and wait for notification of job termination. Security configuration and registration are not applicable.

The attributes of interest are the scenario steps and non-functional characteristics as follows:

- Deployment Infrastructure (dependence on)
- Security configuration (adequacy, tool support, scalability for increasing nodes, services, users)
- Deployment validation
- Host discovery
- Deployment (remote, impact on production container)
- Service registration (adequacy, ease of use)
- Redeployment and Un-deployment (ease of, and impact on production container)
- Non-functional deployment characteristics (scalable, repeatable, ease/speed, traceable, debuggable)
- Deployment security
- Provisioning and QoS for deployed services.

4.2 Evaluation Summary

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 to ensure non-portable applications are resourced appropriately in a heterogeneous environment. There is no capability for "application" registration, no explicit deployment packaging, or valida-

tion of the deployment prior to use. There is an explicit security model for deployment (just the MMJFS security settings), which is therefore identical for deployment and job submission across 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 we suspect that 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). In theory it would be possible 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 default 1st order service security model is not very scalable, requiring deployment time 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 security model may be possible which would be more scalable. Deployment allows for immediate invocation but does not guarantee QoS.

4.3 Analysis

Table 3 Deployment evaluation

Attributes	**1st order**	**2nd order**
Deployment infrastructure	N/A	-0.8
Security configuration	0	N/A
Validation	0	-1
Host discovery	1	1
Deployment	-0.5	1
Service registration	0.5	N/A
Redeployment and un-deployment	-1	0.5
Non-functional characteristics	-1	1
Deployment security	-0.5	0.5
Provisioning	0	1
Overall	**-0.17**	**0.4**

From Table 3 it is clear that the 2nd order approach provides better support for remote, repeatable, scalable deployment of *applications*. Significant infrastructure is required as a precondition for 2nd order deployment, but we assume no specific deployment infrastructure for the 1st order approach but only local manual deployment. No security configuration or service registration is applicable (or indeed possible) for the 2nd order approach. Deployment validation is not supported by either approach, although it is possible in theory with the 1st order approach (but not for 2nd order given the absence of explicit deployment packaging). The 1st order approach supports service registration. Host discovery is supported by both approaches. The 2nd order approach

supports deployment, redeployment, deployment security, and provisioning/QoS, and demonstrates good non-functional deployment characteristics such as scalability, repeatability and traceability. Overall, the 2nd order approach contributes positively to deployability (0.4) compared with the 1st order approach (-0.17). For a more detailed treatment of deployment and remote deployment experiment results using a 3rd party technology see [7]. It is possible to provide for remote deployment of services in OGSA and this functionality is expected to be included in GT4 [28].

5 Run-time Use

5.1 Scenario and Attributes

A number of different (and probably competing) non-functional run-time goals for a Grid infrastructure are possible. These include minimizing the response time for a job for a user, maximizing scalability/throughput for one or more users, ensuring fairness of usage, and preventing saturation of resources. There are also different types of jobs – interactive (short-running), and batch jobs (long-running) – which correspond to the typical uses of the 1st and 2nd order approaches. Other important attributes include availability, reliability and stability. The experiments we conducted were based on GTMark, a new Grid benchmark which can flexibly emulate various types of jobs and loads, and measure and report non-functional metrics [6]. We configured it to emulate a work-flow based load, calling short stateful services for each task (taking approximately 60s on the slowest machines) with 10s of MB of data passed each time.

We planned to experimentally compare the 1st and 2nd order approaches using the benchmark but did not adopt this methodology for theoretical and practical reasons. Practically this was due to difficulties installing and configuring the "All Services" version of GT3.2 on all sites, developing the 2nd order version of the benchmark, and bugs in critical parts of the infrastructure (E.g. SOAP attachments). More fundamentally we determined that the use of the benchmark introduced artificial artefacts (related to starting a new JVM for each task) which would inevitably skew the results making a direct comparison using GTMark largely uninformative. The following analysis was therefore conducted analytically although informed by experimental results [6, 8, 9, 10] and, and must therefore be understood to be primarily qualitative.

The run-time mechanisms used for the 1st order approach are container, GWSDL, SOAP server, SOAP attachments, MDS3, and load balancer (optional, we used client-side load balancing). The mechanisms used for the 2nd order approach are container, SOAP server, MMJFS, RSL-2, MDS3, data transfer services, and resource scheduler (optional). The 1st order service use task scenario is discover services, determine application resource/service requirements (e.g. all the services required for an application, maximum concurrency, etc), select services that are appropriate and available to the user, determine how to invoke services (i.e. obtain and inspect WSDL, generate stubs), create (or request) required number of service instances on selected resources, transfer input data, call service(s), and obtain result data. The 2nd order scenario is

simpler. Assuming that the client has obtained a handle to the MJS instance already (as a result of the deployment phase) and that the job has been submitted to the resource scheduler the client just waits for the job to be run and a result generated.

The attributes of interest are a combination of the scenario steps, and non-functional characteristics as follows:

- Service discovery (discovering services which are available to a user, and dynamically binding to them)
- Service resourcing (availability/timeliness, QoS guarantee, load balancing, fairness of resource sharing, prevention of saturation, failover)
- Performance and Scalability (impact of security, overhead compared with local call, performance, scalability, hyper-threading scalability, initial call overhead, instance creation/destruction, scalability limits, long running jobs, stability)
- Reliability and Availability (reliability, recovery and exception handling, availability)
- MDS3 performance (performance, scalability, outstanding performance issues)
- Data transfer (functionality, scalability, experimental results).

5.2 Evaluation Summary

The 1st order approach supports service discovery at run-time time. However, the availability (either in terms of permission to use the service, or availability of resources for a given QoS) may not be guaranteed. There is also no default support for service resource management and load balancing or for failover. Measured performance and scalability were excellent. In particular, there is minimal overhead in using GT3, or using security, and scalability was excellent for increasing load up to saturation, in the saturated region, and for hyper-threading CPUs. Long-term stability was good over a 12 hour period. However, performance for the first use of an stateful instance, or for calls to stateless instances, may be slightly worse, there may be issues with scalability of instance creation and destruction, and there are limits to scalability and response times due to the use of blocking synchronous calls. Reliability is good, but there are likely to be issues with availability due to the possibility of resource saturation. The index service performance and scalability were good, but there may be outstanding issues with performance, scalability, liveness, and consistency. We were unable to get a production version of data transfer with SOAP attachments working due to problems with both Axis and GT3. Tracing and debugging service use with security turned on is problematic, and needs container based support, as proxy approaches do not work with stateful instances.

The 2nd order approach uses MMJFS and will therefore suffer from unpredictable, and possibly extended delays between job submission and execution. Performance and scalability are also poor due to the architecture, implementation, and high resource usage. There is support for fault-tolerance and for the use of 3rd party resource management products providing a range of policies. We were unable to get data transfer working across sites. Security of use and deployment are identical although

GRIM may provide more sophisticated (but delayed) run-time security. The 2nd order approach is unsuitable for running large numbers of short, independent, or interactive jobs, or for jobs which require a complex client or orchestration. RSL-2 files must be customized (e.g. for job concurrency requirements, or for splitting a job over a number of separate resources) and passed to MMJFS at run-time.

5.3 Analysis

Table 4 Run-time Use evaluation

Attributes	1st order	2nd order
Service discovery	0.333	N/A
Service resourcing	0.333	0.5
Performance and Scalability	0.5	-0.166
Reliability and Availability	0.333	0.666
MDS3 performance	0.5	N/A
Data transfer	0.333	0.333
Overall	**0.39**	**0.333**

Table 4 shows that the 1st order mechanisms all contribute although none are perfect and the highest, performance and scalability (container, services, and MDS3) are only 0.5. For the 2nd order approach, service discovery and MDS3 performance are not applicable, performance and scalability hinders, and data transfer is neutral. However, resourcing and reliability and availability are positive. Comparing the two approaches, service discovery and performance are supported by the 1st order approach, but not for the 2nd order approach. Service resourcing and reliability/availability are better for 2nd order approach. Performance and scalability and data transfer are better for 1st order approach. Although the 1st order approach supports service discovery, it may not be possible to search for available services (which a user has permission to use, or for which adequate resources are currently available - although services can be invoked immediately after discovery). Using the 2nd order approach, we assume that a MJS handle is passed back to a client as a result of MMJFS deployment, so service discovery is not applicable. The 2nd order resource management model is more sophisticated than the 1st order approach, but does not provide support for interactive short jobs as there may be long delays waiting for the use of resources which have been booked in advance or currently locked by long running jobs. The 1st order approach on the other hand needs better support for fair sharing of resources, load-balancing across heterogeneous resources, prevention of resource saturation, provision of QoS, and failover.

Nevertheless, performance and scalability of the 1st order approach is better, with minimal impact of the GT3 framework or security, and excellent measured performance and scalability. However, we note that there may be issues with the scalability and performance of instance creation and destruction, the likely existence of a scalability threshold (per container or machine), and problems with timeouts on long running services. In contrast, the 2nd order approach imposes a high performance overheard and scalability will be poor. Reliability is good for both approaches, but surprisingly there are problems with recovery/exception handling and availability

with the 1st order approach (although these could be reduced by using multiple sand-boxed containers, and attention to safe client side programming practices). The 2nd order approach has good fault-tolerance. Performance and scalability of the index service are good, but there are possibly outstanding issues. Data transfer using SOAP attachments should work well in theory, but in practice there were problems, including the interaction of security and attachments. However, the 2nd order data transfer services could not be made to work at all. Overall the 1st order mechanisms contribute to Use (0.39) only slightly more than 2nd order (0.333), but there is substantial room for improvement for both.

6 Comparison and Predicted Improvement

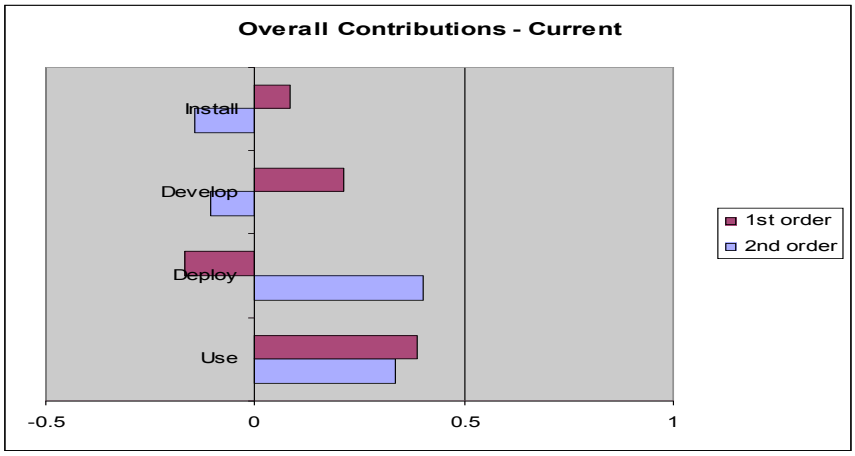

Figure 3: Overall Contributions - Current

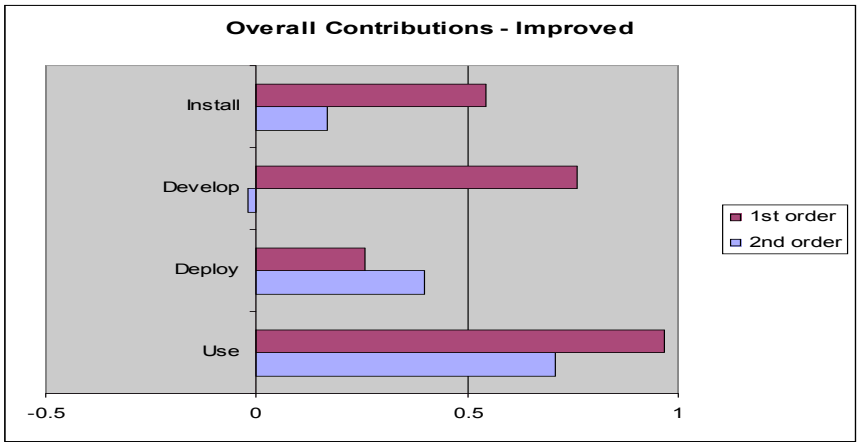

Figure 4: Overall Contributions - Improved

Figure 3 shows that both approaches currently contribute to Use, 1st order contributes to Development, and 2nd order contributes to Deployment. The 2nd order approach is better than 1st for Deployment, 1st order Development is better than 2nd, and 1st order Installation is only marginally better than 2nd order.

Assuming that improvements to mechanisms with neutral contributions to primitive attributes (for details, see Appendix and [4]) are possible (whereas, mechanisms with negative contributions can not be as easily improved as they are caused by problems such as architectural mismatch rather than purely implementation issues), Figure 4 shows the overall predicted quality improvements. 2nd order Use is improved, Deployment is unchanged, but Development and Installation still have weaknesses. 1st order Use, Development and Installation are significantly better, Deployment is improved, but Deployment and Installation have further potential for improvement. It is conceivable that GT4, the current version of the Globus Toolkit, is likely to reflect some of these improvements.

7 Alternative Approaches

We now explore some alternative architectures and technologies from the boundaries of component, web and grid services, which have potential for bridging the two ways of using SOAs.

Service mobility. The "Are web services distributed object systems?" debate [16] is relevant to the Grid. There are a number of Grid requirements such as service lifecycle management and deployment flexibility which suggest that the answer to this question, at least for Grid Services, is "yes". Grid deployment requirements can be viewed as ranging across a spectrum from "cold" through "warm" to "hot" deployment, which encompasses part of the mobile object spectrum. In some cases it will be desirable to move components from one resource to another, even while they

are in the process of running (typically to cope with failures or demand/capacity fluctuations, or to optimize resource availability). There are therefore good reasons to relate deployment to lifecycle management issues, including service mobility, and other non-functional characteristics such as resource management and failover. The 1st order approach provides at least the possibility of writing services whose persistence and activation/passivation is managed by the container. However, integration of these mechanisms with a cross-organizational deployment capability (e.g. to reactivate passivated services on nodes across firewalls) would be a major hurdle, although some research has started to address this [16, 17, 18].

Reducing deployment frequency. In the absence of automated remote deployment, another tactic is to attempt to reduce the number of times the deployment scenario is enacted. The deployment scenario must be repeated for a service when its implementation is changed, so if the implementation could be changed without having to repeat the full deployment process for all Grid nodes some effort could be saved. This approach splits the deployment role into two phases: initial deployment; and maintenance. At least two approaches are possible.

The first approach is to split the implementation of the service into two parts: the main service with the (hopefully invariant) service interface description deployed to multiple resources once only, and the part liable to change encapsulated as a service on the developer's machine. This enables the developer to change it locally, with the changes picked up automatically by the remotely deployed services. This is essentially the Strategy design pattern for web services [19]. Can this be made to work for Grid services? The main service just invokes the appropriate strategy as an external service. However, in the Grid version the strategy part is only resourced by the machine it is deployed on so it will not scale unless it is deployed to multiple machines, introducing the deployment problem again. Another side-effect is the increased possibility of run-time errors.

What is required is the ability to take a copy of the strategy code and run it on the same resources as the main service, thus bringing us back to the realms of distributed objects and object mobility. Using SOAP attachments it would be possible for the strategy service to supply a run-time Java object to the main service (at instance creation time) which would then be resourced by the same container. A variation on this approach is for the client to provide the algorithm directly to the Factory Service (or even the service instance) at invocation time, either in source form in the SOAP body, or as a Java run-time object as a SOAP attachment.

The second approach is a hybrid inversion of this using 1st order services and MMJFS. A 1st order service is deployed to a single resource, and acts as the front-end entry-point service, which then passes an executable object to MMJFS for resourcing, waits for the result, and then passes it back to the calling client. The client only interacts with the 1st order service, and to all extents and purposes it looks like a real service, with a real GWSDL description. Because the resourcing is handled with MMJFS only one deployment of the front-end service is required (assuming that it is deployed to a machine with sufficient resources). However, the development and packaging of the service would need to allow the division between interface, executable, and job submission, and poor performance for interactive, short-running jobs using MMJFS

would be serious. This approach would inevitably introduce unacceptable development and run-time overheads.

Resourcing 1st order services. A number of projects have investigated ways of integrating OGSA services with resource managers including exposing Condor services as OGSA services [11], using a trading service [12], and integrating a BPEL engine with the Sun Grid Engine (SGE) to act as a resource broker between the BPEL process and partner services [13]. These studies typically conclude that the integration is more difficult than expected, exception handling is problematic, and that a production quality resource management system must be used otherwise scalability and fairness/accountability suffer. There is no obvious solution for the problem of poor performance and latency due to the insertion of a heavy-weight resource management system in OGSA. However, other research to watch includes the analysis of component performance and performance-aware middleware [14], and the use of services to dynamically deploy and host services on Grids or the internet [29], which replaces the notion of "jobs" with a clear separation between Service Providers (who offer services), and Host Providers (who offer resources).

One approach suggested by the Globus team (but not tried in the evaluation, and possibly no longer directly supported in GT4), is to use a central Virtual Factory Service to decide where to create new instances. This would basically function as a global resource broker for instance management, but still requires the service to be deployed in advance to all the machines. In GT4 it may be that WS-RF, WS-Routing, WS-Addressing and WS-Agreement will provide the basis for a solution [15].

Reducing deployment effort: "Zero-effort" deployment. A different approach is to reduce the effort for development and deployment of components using light-weight containers. 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 [20]. Also related is Inversion of Control (IoC) and Aspect Oriented Programming (AOP) approaches to component portability, giving the ability to separate concerns, such as configuration from use [21, 22]. The core concept behind IoC is that a component exposes its dependencies (anything that a component needs to work such as implementation, other components and services, and even system resources) externally. Encapsulation is relaxed so that any dependency that is managed (or has the potential to be managed) across component boundaries is exposed for higher management. In a nested graph, each component in the call chain exposes its dependencies to the outer caller that uses it, who in turn exposes those dependencies including any of its own to its caller and so on, until all dependencies appear at the top. The top level object then assembles the dependency graph before activating the components, and acts as the "container" and entry point into the system.

Automated Java clustering is a recent application of IoC/AOP [23]. IoC has been applied to Grid Portals to a limited extent, for example GridPort and GridSphere [24, 25]. There is interesting early work on Aspect Oriented WSDL (AO-WSDL) [30]. Because of the significant intersection of the solution and problem spaces, there is potential for the adaptation and application of the IoC pattern more widely to Grid

service deployment, configuration, resourcing and lifecycle management *by services*. Depending on the IoC approaches used, and the extent to which they can be extended to services, potential benefits include: flexible deployment (ranging from lazy/just-in-time deployment to complete pre-deployment and validation), separation of deployment of interfaces and factories from deployment of implementation (giving the capability to store implementations in distributed code repositories, and provide for dynamic provisioning), the ability to modify service behaviour without changing services (for example, using interceptors for logging, tracing and debugging), and the ability to construct light-weight customised (and possibly virtual or distributed) containers containing all (but only) the components and services required, with precise resourcing.

Automating deployment. Given the lack of support for automated remote deployment or infrastructure and services in the Globus middleware stack, and our observations resulting from the evaluation that it is feasible to improve support for remote deployment, we trialed the use of a 3rd party component deployment technology (SmartFrog [26]) for Grid deployment. We conducted five experiments [6, 7] and summarise the results here.

The initial solution worked well in the laboratory, but relied on a number of assumptions that proved impossible to obtain across organisational boundaries. The most significant issues were configuration of the deployment infrastructure outside the laboratory setting, opening a separate port for SmartFrog (given the perception that it is a perfect virus propagation mechanism), and securing the deployment infrastructure. SmartFrog and Globus use different security models. In order to deploy infrastructure securely with SmartFrog an independent (and redundant) security infrastructure, process, and certificates is required which introduces another layer of complexity into already complex infrastructure and security environments.

Another 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 which map user certificates to local accounts) for each site. 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 ultimate goal of deploying secured *services* to an already deployed infrastructure was unachievable, as service specific security configuration is not scalable, and due to the lack of “hot” deployment in GT3, stopping and restarting a production container is unlikely to be acceptable.

From these experiments we conclude that bootstrapping one infrastructure with another is complicated as there are complex interactions between different security models, and debugging deployment infrastructure and deployment processes is difficult, particularly in the presence of security. Installing, configuring, and securing a separate, redundant, and possibly even incompatible deployment infrastructure is non-trivial, and it would be preferable for deployment to be supported as a first-class activity by the Grid middleware stack.

8 Conclusions

Finally we turn to three related but progressively less demanding questions. Will the two approaches converge? Can they be combined, for example, in the one application? And, are they compatible on shared resources?

Convergent. First we consider if the two approaches are likely to converge, even if standards converge (E.g. GT4 is now based on a Web Services standard, WS-RF. [31] addresses the impact of WS-RF on grid). The requirements for 1st and 2nd order approaches are significantly different. For example, the security model (and therefore infrastructure and configuration) required to run arbitrary code possibly as privileged users, is different to that required to run services implemented as components and hosted in sand-boxed containers. Also, if service portability is assumed for 1st order services, but not for legacy code, then the ability of the resource manager to match jobs to appropriate resources is critical.

At least interchangeability between services and applications is required for convergence. Services must be resourced on an equal footing with applications, and applications must be accessible as 1st order services (and support explicit lifecycle steps). Convergence requires significant changes in infrastructure architecture and services, including secure automatic scalable deployment of services, or applications as services, integrated with a resourcing model which guarantees QoS for deployed services and applications, and support for multiple interaction protocols (for example, to abstract the difference between data transfer approaches). Convergence therefore requires the union of the most demanding requirements for each approach to be supported uniformly across the infrastructure, which will result in infrastructure which is complex, monolithic, and expensive to install and maintain.

Combinable. Next we investigate if the two approaches are interoperable, or combinable. For example, can applications be built that use both approaches together? For static development time binding of client applications or work-flows to services or applications, there is no theoretical impediment to using 1st order services, and applications via MMJFS, in the same client. However, the interaction and data transfer models used between client and MMJFS, and client and 1st order service are different, making development, deployment, and use more complicated. Support for uniform dynamic run-time discovery and binding is missing although there may be workarounds using bridging or proxy technologies. The installation and security requirements are likely to be the worst of both worlds.

Compatible. The compatibility of the two approaches raises a number of questions. Can applications, services and users running the two approaches independently of each other be supported on the same resources? Can resource management work fairly for both approaches using the same resources? What is the impact on container performance of running MMJFS in the same container as 1st order services? What is the interaction of security settings across multiple installations? What is impact on availability and performance of hosting both approaches together?

One fundamental problem with using 1st and 2nd order approaches on the same infrastructure is resource use and sharing. By default a 1st order service can be invoked and will consume resources resulting in rapid saturation of resources in the absence of any mechanism to limit resource consumption (and also making the system open to denial of service attacks). Because 1st and 2nd order approaches use different resourcing models there is currently no default way of ensuring equitable sharing of resources between the two approaches. Some resource managers are exclusive and lock resources for long periods of time while legacy jobs are running on them, thereby precluding interactive short-term jobs from being resourced in a timely fashion. On the other hand 1st order services may get preferential treatment to 2nd order jobs, as they are automatically and opportunistically resourced once deployed, making it harder for 2nd order jobs to get adequate resources for long enough periods of time to complete (there may be high costs associated with stopping and restarting 2nd order jobs). Requiring all resourcing to be managed by a heavy-weight 2nd order resource manager is likely to be expensive in terms of resource overheads, and severely impact the performance of 1st order jobs. It may be possible to build centralised resource managers that enforce/coordinate resource management policies using appropriate (and different) mechanisms for 1st and 2nd order services. For example, 1st order services could be resourced by containers, which enforce resource usage (e.g. using passivation/activation) with reference to external policies. Another approach (used for Web farms) may be to apportion the resources between different categories of users or jobs.

We have noted interference of security configurations across multiple versions of infrastructure installed on shared resources, configured for different uses [25]. The challenge is isolating the security configuration of multiple installations for servers and clients. Given that it is unlikely that one container will support both approaches from a security perspective, the likelihood of MMJFS impacting the performance of 1st order services, and the lack of "hot" deployment for services, multiple containers will be required to ensure scalability and isolation, and to make it easier to configure, monitor and assign appropriate resources to support each approach.

This evaluation demonstrates that the set of mechanisms used for the 1st and 2nd order approaches to Grid SOAs contribute in different ways to Installability, Developability, Deployability, and Usability. Neither approach was perfectly realized in GT3.2 and impediments exist to uniform incremental improvement. There are areas that are easier to improve than others and we expect future instantiations of OGSA such as GT4 to gain from these. However, convergence, combination, and compatibility are not trivially achieved. We conclude that there is more research to be done in order to use Web Services to build, use, and maintain large scale, production quality Grid infrastructures for disparate uses, and supporting multiple cross-cutting roles across organizational boundaries.

Acknowledgements

Thanks to Len Bass for discussions which helped to refine our architectural analysis methodology, and to Ian Foster for feedback on the draft version of the technical

report [4]. The empirical component of this work was done as part of the UK OGSA evaluation project funded by the EPSRC grant GR/S78346/01, while the analytical analysis was supported by the CSIRO ICT Centre.

Appendix: Detailed Attribute Contributions

Task: Step/quality attribute	1st Order Contribution	2nd Order Contribution
Installation	0.08	-0.15
Core package	0.625	0.625
Package	1	1
Independent	1	1
Portable	1	1
Build	N/A	N/A
Repeatable and reliable build	N/A	N/A
Configure	0	0
Test	1	1
Scalable	1	1
Manage	0	0
Remote manage	0	0
Security	-0.5	-0.5
Package	0	0
Independent	-1	-1
Portable	-1	-1
Build	0	0
Repeatable and reliable build	0	0
Configure	-1	-1
Test	-1	-1
Scalable	-1	-1
Manage	0	0
Remote manage	0	0
MDS3	0.5	0.5
Package	1	1
Independent	1	1
Portable	1	1
Build	N/A	N/A
Repeatable and reliable build	N/A	N/A
Configure	-1	-1
Test	1	1
Scalable	1	1
Manage	0	0
Remote manage	0	0

Client side installation	-0.285	-0.555
Package	-1	-1
Independent	0	-1
Portable	0	-1
Build	N/A	0
Repeatable and reliable build	N/A	0
Configure	-1	-1
Test	0	-1
Scalable	0	0
Manage	0	0
Remote manage	N/A	N/A
All Services package	N/A	-0.8
Package	N/A	1
Independent	N/A	-1
Portable	N/A	-1
Build	N/A	-1
Repeatable and reliable build	N/A	-1
Configure	N/A	-1
Test	N/A	-1
Scalable	N/A	-1
Manage	N/A	-1
Remote manage	N/A	-1
Develop	0.21	-0.1
Step - Implement Service	0.428	0.333
Explicit Component Model	1	0
Run-time support	1	N/A
Container managed services	0	N/A
Service types	1	N/A
Component safety	-1	N/A
Portable	1	0
Legacy/binary code	0	1
Step - Description: Interface definition	0.333	-0.333
Interface definition - expressive	1	1
Interface definition - ease of use	0	-1
Interface definition - standard	0	-1
Step - configuration	0.5	N/A
Deployment descriptor - expressive	1	N/A
Deployment descriptor - ease of use	0	N/A
Deployment descriptor - standard	1	N/A
Step - packaging	0.333	N/A
GAR file - expressive	1	N/A
GAR file - ease of use	0	N/A
GAR file - standard	0	N/A

Step - Resource requirements	N/A	0.333
Resource requirements - expressive	N/A	1
Resource requirements - ease of use	N/A	0
Resource requirements - standard	N/A	0
Step - client development	0	-0.75
Client side package	0	0
Declarative security settings	-1	-1
Independent	0	-1
Support for "rich" clients	1	-1
Attribute - Tool support	-0.333	N/A
Development environment - adequacy	0	N/A
Development environment - independence	0	N/A
Development environment - standard/ease of use	-1	N/A
Deploy	-0.17	0.4
Step - Install Deployment infrastructure	N/A	-0.8
Deployment infrastructure	N/A	-0.8
Step - Security configuration	0	N/A
Service specific - adequacy	1	N/A
Tool support	0	N/A
Scalability	-1	N/A
Step - deployment validation and tracing	0	-1
Deployment validation	0	-1
Step - host discovery	1	1
Host discovery	1	1
Step - deployment	-0.5	1
Deployment - Remote deployment support	0	1
Deployment - Impact on production container	-1	N/A
Step - service registration	0.5	N/A
Service registration - Adequacy	1	N/A
Service registration - Ease of use	0	N/A
Step - redeployment and undeployment	-1	0.5
Ease of redeployment	-1	0.5
Attribute - scalability and repeatability	-1	1
Scalable, repeatable, performance, traceable and debuggable	-1	1
Attribute - deployment security	-0.5	0.5
Deployment security	0	1
Validation	-1	0
Attribute - provisioning and QoS	0	1
Provisioning and QoS	0	1
Run-time Use	0.39	0.333
Step - Service discovery	0.333	N/A

Discover service	1	N/A
Service available to user	0	N/A
Dynamic binding	0	N/A
Attribute - Service resourcing	0.333	0.5
Availability	1	1
QoS guarantee	0	1
Load balancing	0	0
Fairness/resource sharing	1	-1
Saturation prevention	0	1
Failover	0	1
Attribute - Performance and Scalability	0.5	-0.166
Impact of security	1	N/A
Overhead compared with local call	1	-1
Performance	1	0
Scalability	1	0
Scalability for hyper-threading	1	0
Initial instance creation overhead	0	N/A
Scalabiity of instance creation/destruction	0	N/A
Scalability limit	0	-1
Long running jobs	-1	1
Stability	1	N/A
Attribute - Reliability and Availability	0.333	0.666
Reliability	1	1
Recovery - Exception handling	0	1
Availability/Timeliness	0	0
Attribute - MDS3 Performance	0.5	N/A
Performance and Scalability	1	N/A
Outstanding Issues	0	N/A
Attribute - Data transfer	0.333	0.333
Functionality	1	0
Scalability	0	1
Test results	0	0

References

1. Schulze, B., Madeira, E.R.M., Contracting and Moving Agents in Distributed Applications Based on a Service Oriented Architecture. First International Workshop on Mobile Agents (1997). Springer LNCS Volume 1219. 74-85.
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. The UK-OGSA Evaluation Project. http://sse.cs.ucl.ac.uk/UK-OGSA/

4. Brebner, P., Two Ways to Grid: Service-centric vs. Resource-centric evaluation of the Open Grid Services Architecture (OGSA), CSIRO ICT Centre Technical Report (2005). http://www.ict.csiro.au/staff/paul.brebner//TwoWaysToGrid.htm
5. S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maguire, T. Sandholm, P. Vanderbilt, D. Snelling: Open Grid Services Infrastructure (OGSI) Version 1.0. Global Grid Forum Draft Recommendation (2003)
6. 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
7. Brebner, P., Emmerich, W., Deployment of Infrastructure and Services in the Open Grid Services Architecture (OGSA), submitted to CD2005 (2005).
8. GT3.0 Performance and Scalability Tests. http://www-unix.globus.org/ogsa/tests/scalability/single-index.html
9. Chen, D., et. al., OGSA Globus Toolkit 3 Evaluation Activity at CERN. http://www2.twgrid.org/CERN_news/osgc-2.pdf
10. Xiao-Dong Wang, and Rob Allan, GT3.2 Beta MMJFS Test Report, Technical Report, e-Science Centre, CCLRC Daresbury Laboratory, (2004). http://esc.dl.ac.uk/Testbed/mmjfs.pdf
11. Chapman, C., et. al., Condor Services for the Global Grid: Interoperability between Condor and OGSA. UK e-Science All Hands Meeting (2004). http://www.cs.ucl.ac.uk/staff/w.emmerich/publications/AHM04/condor-ws.pdf
12. Butchart, B., Chapman, C., Emmerich, W., OGSA First Impressions – A Case Study in Re-engineering a Scientific Application with the Open Grid Services Architecture. UK e-Science All Hands Meeting (2003). http://www.cs.ucl.ac.uk/staff/w.emmerich/publications/AllHands03/ogsa.pdf
13. Nowell, H., Butchart, B., Coombes, D., Price, S., Emmerich, W., Catlow, C., Increasing the Scope for Polymorph Prediction using e-Science. UK e-Science All Hands Meeting (2004). http://www.cs.ucl.ac.uk/staff/w.emmerich/publications/AHM04/finalnowell.pdf
14. McGough, S., Young, L., Afzal, A., Newhouse, S., Darlington, J., Performance Architecture within ICENI. UK e-Science All Hands Meeting (2004). http://www.lesc.ic.ac.uk/iceni/pdf/ahm2004_performance.pdf
15. WS-Agreement. http://www.gridforum.org/Meetings/GGF11/Documents/draft-ggf-graap-agreement.pdf
16. Vogels, W., Web services are not distributed objects. IEEE Internet Computing Vol. 7, No. 6 (2003). 59-66. http://doi.ieeecomputersociety.org/10.1109/MIC.2003.1250585
17. Weissman, J., Kim, S., England D., Supporting the Dynamic Grid Service Lifecycle (2004). http://www-users.cs.umn.edu/~jon/papers/cs2004.pdf
18. Weissman, J., Kim, S., England D., A framework for dynamic service adaptation in the Grid (2005). http://www-users.cs.umn.edu/~jon/papers/ngs2005.pdf
19. Observer and Strategy patterns for Web Services. http://www-128.ibm.com/developerworks/webservices/library/ws-dbarch/?Open&ca=daw-ws
20. Matthew, S.: Examining the Validity of Inversion of Control. The Server Side. (2005). http://stage.theserverside.com/articles/article.tss?l=IOCandEJB
21. Fowler, M.: Inversion of Control Containers and the Dependency Injection Pattern. (2004). http://www.martinfowler.com/articles/injection.html
22. Spille, M.: Inversion of Control Containers. (2004). http://www.pyrasun.com/mike/mt/archives/2004/11/06/15.46.14/index.html
23. Terracotta. http://www.theserverside.com/news/thread.tss?thread_id=34294
24. The GridPort project. http://gridport.net
25. The gridsphere portal framework. http://www.gridsphere.org
26. 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
27. 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
28. Personal communication (email) from Ian Foster 16/06/2005. See also: Foster, I., A Globus Primer. http://www.globus.org/toolkit/docs/4.0/key/GT4_Primer_0.6.pdf
29. Watson, P., Project Dynasoar: Dynamic Deployment of Web Services on a grid or the Internet. http://www.neresc.ac.uk/projects/dynasoar/
30. Grundy, J.C., Panas, T., Singh, S., Stoeckle, H. An Approach to Developing Web Services with Aspect oriented Component Engineering, In Proceedings of the 2nd Nordic Conference on Web Services, 2003. http://www.cs.auckland.ac.nz/~john-g/papers/ncws2003.pdf
31. Joseph, J., Ernest, M., Fellenstein, C., Evolution of grid computing architecture and grid adoption models.IBM Systems Journal, Volumn 43, Number 4 (2004). http://www.research.ibm.com/journal/sj/434/joseph.html