Software, Component, and Service Deployment in Computational Grids

Gregor von Laszewski[1], Eric Blau[1], Michael Bletzinger[2], Jarek Gawor[1], Peter Lane[1], Stuart Martin[1], and Michael Russell[1]

[1] Argonne National Laboratory, 9700 S. Cass Ave., Argonne, IL 60439
[2] National Center for Supercomputing Applications, Champaign, IL 61821,
gregor@mcs.anl.gov

Abstract. Grids comprise an infrastructure that enables scientists to use a diverse set of distributed remote services and resources as part of complex scientific problem-solving processes. We analyze some of the challenges involved in deploying software and components transparently in Grids. We report on three practical solutions used by the Globus Project. Lessons learned from this experience lead us to believe that it is necessary to support a variety of software and component deployment strategies. These strategies are based on the hosting environment.

1 Introduction

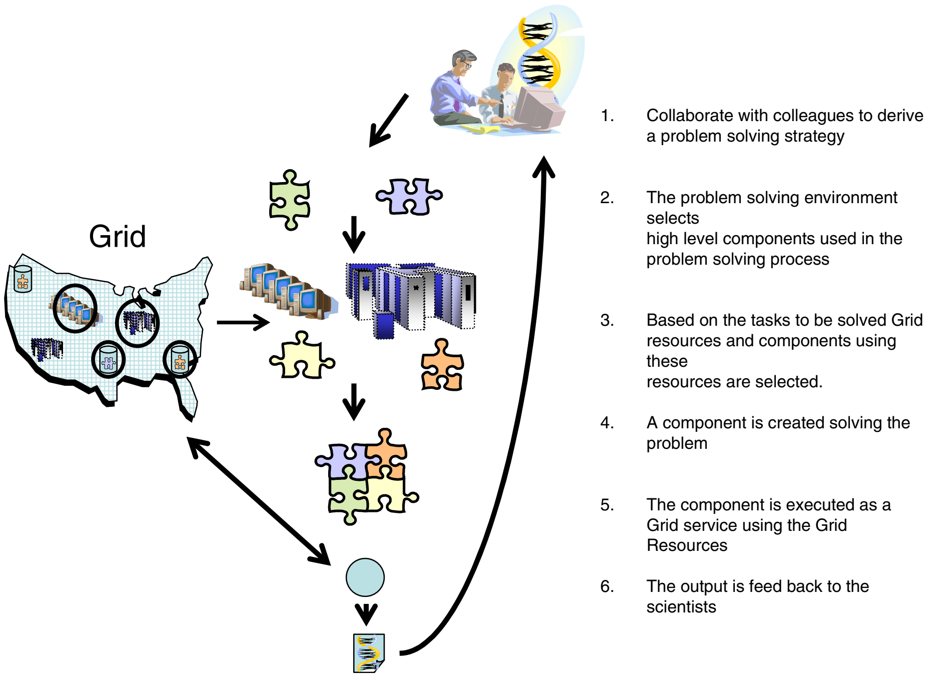
Grids comprise an infrastructure enabling scientists to use a diverse set of distributed software, services, and components that access a variety of dispersed resources as part of complex scientific problem-solving. This infrastructure includes the use of compute resources such as personal computers, workstations, and supercomputers; access to information resources such as directory services and large-scale data bases; and access to knowledge resources such as collaboration with colleagues. A central role in defining Grids is the creation of virtual organizations that define sharing and trust relations between the diversified set of resources. Deployment of software, components, and services must be governed by the appropriate definition of rules and policies. Such sharing rules may be rather simple, as demonstrated by the SETI@home project [10] to allow the creation of commodity compute resources pools. The resources are contributed by a large number of individuals. It is important to recognize that providing an easy deployment strategy with easy-to-understand rules of engagement results in integration of resources that can be provided by nonexperts. More complex rules are defined as part of virtual organizations spanning resources among traditional compute centers. They enable access to high-end resources (such as supercomputers) and advanced instruments (such as a particle collider). Examples are the DOE Science Grid [2], the NASA Information Power Grid (IPG), and the Alliance Virtual Machine Room (VMR) [14,1]. Sharing rules govern the privileged use of resources contributed by the centers. The deployment of software, services, and components in such production Grids is performed by experts and well-trained administrative staff following guidelines set within and between such compute centers. In each case it is important to develop proper deployment strategies as part of every Grid-enabled infrastructure.

In the rest of the paper we first present a simple example that introduces several issues that must be addressed while deploying software, components, and services in Grids. Based on our requirements analysis, we have identified three scenarios that allow us to deploy components in a Grid-based infrastructure. We compare these scenarios and present our conclusions.

2 Example

We present a simple example illustrating a small set of requirements that we must deal with. In Figure 1 a group of scientists needs to deploy a problem-solving environment on various resources in the Grid to conduct a large matrix factorization as part of the structure determination of a molecule in three-dimensional space.

Initially, we need to identify a set of suitable resources to perform our task. Often we will identify resources that have the potential to perform a given task but may not have the necessary software deployed on them. We can proceed in



1. Collaborate with colleagues to derive a problem solving strategy
2. The problem solving environment selects high level components used in the problem solving process
3. Based on the tasks to be solved Grid resources and components using these resources are selected.
4. A component is created solving the problem
5. The component is executed as a Grid service using the Grid Resources
6. The output is feed back to the scientists

Fig. 1. Component deployment and assembly to support the scientific problem-solving process in Grids.

one of two ways: eliminating the resource because its software environment is insufficient for the task, or installing the appropriate software to provide the appropriate functionality. Once our environment is present, we access a set of Grid services allowing the composition of a Grid-enabled application based on Grid services and components. For the resources of choice we determine appropriate components suitable for performing our requested task. These components are assembled and executed in order to deliver feedback to the scientists. We emphasize that the scientists do not have to know the algorithmic details of the solving process, as they are hidden in the components. These details include on which Grid resources the factorization is performed or which algorithm is used. Thus, the Grid is used as a utility that returns the information requested as part of the complex problem solving process.

To make such a Grid-enabled environment possible, we must integrate several locally maintained code repositories into a virtual code repository. From it we must discover, select, and assemble the components best suited for the task. The selection of components is determined by the functional properties, performance, licensing issues, and cost. We must address the issue of whether the components can be trusted to be executed on a resource. We must ensure that the interfaces between components provide functional compatibility and,in some cases, version compatibility in order to engage in the creation of reproducible results. Version control must be strictly enforced not only on the delivery of components but also potentially while using statically or dynamically linked libraries. A discovery process must be designed in such a way that only authenticated users may know the existence or the properties of the components. Moreover, prior to the use of the component, the user must be able to judge its functionality and approve it for inclusion within other components. Smart components acting in behalf of users themselves must be able to perform similar component assembly strategies. This may mean that the same query and assembly of a component for user A could result in a completely different implementation for user B, who may have different access rights to resources within the Grid.

3 Deployment Cycle

As part of our previous example we identified three basic user communities that must be integral part of every deployment strategy: *programmers and designers* that are developing software, components, and services in a collaborative fashion to be deployed in the Grid; *administrators* that deploy them; and *application users* that access deployed software, component, and services. Thus, in order to address the complex issues related to deployment, the software engineering cycle must reflect dealing with this issue in each stage, while at the same time forcing and supporting interactions between designers, administrators, and users.

A blueprint for managing deployment of Grid software, services, and components is depicted in Figure 2. It supports the packaging, deployment, and iterative process of improving the infrastructure. The goal of this rigorous software process is to reduce the effort to deploy and maintain the infrastructure that

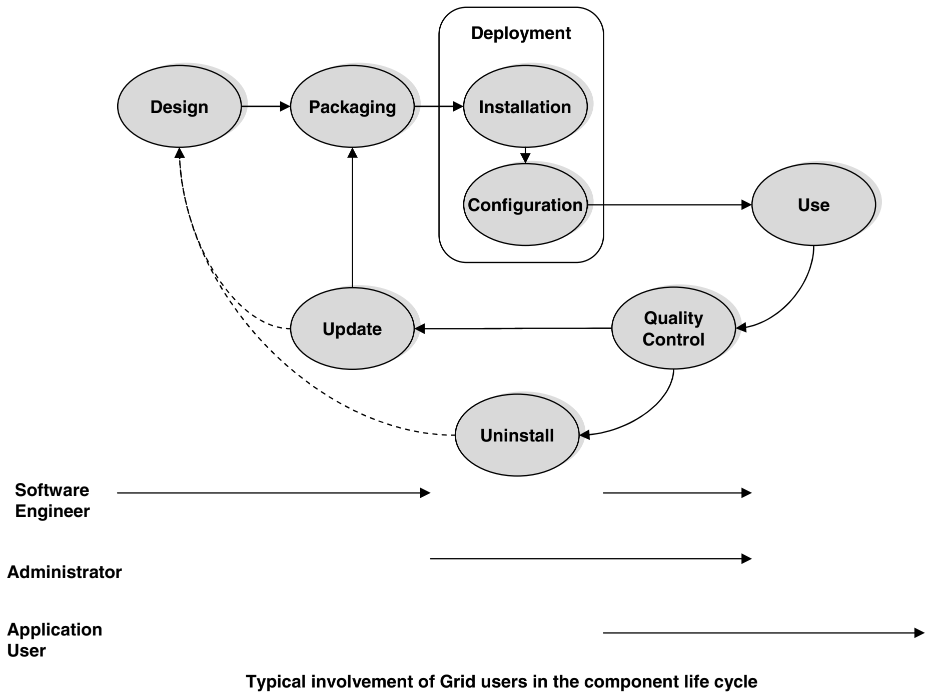


Fig. 2. Deploying and maintaining software, components, and services in Grids is a resource-intensive task that requires an iterative interaction between designers, administrators, and users

enables the effective use of the complex environment by the application users or scientists, and also by other Grid architects developing new Grid services.

So far we have not distinguished between what we understand under software, components, and services because we believe that our observations are applicable to each one of these categories. In order to deploy components and services in Grids, we need to prepare what the Grid community sometimes calls a hosting environment. Such hosting environments provide the necessary software infrastructure for executing Grid-related services and components. The the Java Virtual machine, J2EE, .NET, a prepackaged Grid hosting environment [1], or simply a particular version of an operating system are examples. In order to prepare such a hosting environment, we can obviously benefit from software engineering practices that deal with software deployment as it "defines the assembly and maintenance of the resources necessary to use a version of a system at a particular site" [15]. Once we have established a hosting environments, we can develop services [9] that are based on the hosting environment and components that uses the services as part of a component framework. The problems associated with deployment are manifold and not unique to Grids. Nevertheless, common commercial distributed environments such as CORBA [13] provide insufficient support for the deployment in Grids: whereas these technologies usually target a single administrative domain, Grids encompass multiple domains while keeping them autonomous. One of the key issues in a Grid environment must be a security infrastructure that simplifies deployment and use of services and components. A convenient Grid Security Infrastructure (GSI) including single-signon capability has been developed by the Globus Project, which we can use to support the deployment cycle [3].

4 Deployment Scenarios

Deployment of Grid software, components, and services imposes numerous requirements. Hence we cannot assume that a single strategy fulfills all of these requirements. Instead, we concentrate on three common scenarios that we have identified within our example and provide a deployment strategy solution for each of them. We have termed them thick, thin, and slender in analogy to terminology used in the Internet community.

Thick Deployment: A Grid designer develops software enabling services that are installed locally on a computer by an administrator. Such component and services are typically written in C and may be run with root access on the machine. They usually are tightly integrated with an operating system.

Thin Deployment: A scientist is using a Web browser to access Grid Services in a transparent fashion. The communication is performed only through a browser interface and does not allow installation of any software on the client machine on which the browser runs. A thin Grid service may allow the scientist to interface with a thick Grid service.

Slender Deployment: A slender client allows a platform-neutral deployment under the assumption that an appropriate hosting environment is already present (through, for example, the use of a thick service). A good example of such a slender deployment service is the use of Webstart within a JVM hosting environment. It allows the browser to cache components locally, as well as integrate specialized local available applications within the Web interface.

In the next sections, we describe a deployment strategy for each of the scenarios in the Grid.

5 Thick Deployment

Discussed in [4] is a subset of deployment issues for Grid services and components based on traditional programming languages such as C and FORTRAN and applied to the Globus Toolkit. As an initial solution, the Globus Project, together with NCSA, has developed a Grid Packaging Tool (GPT) that simplifies creation and the deployment of precompiled software. The code is prepared with auto configuration tools and is also available as source code. The GPT separates machine architecture probing from the tests done for a single site or machine deployment. This approach allows a component to be precompiled for a certain architecture and then deployed during installation by means of a setup script.

Hence, the only probing needed is a dependency check that makes sure the required dependent packages are available.

Issues that are intended to be addressed with this tool are as follows:

– An intercomponent dependency checker that can track dependencies at compile, link, and runtime.
– A versioning system that identifies compatibility between different versions of components based on a variation of the libtool versioning.
– Distribution of binaries with compile time flavors, such as the numerical resolution.
– Distribution of relocatable binaries that are independent of any absolute path within the distribution.
– Integration of dependencies to external programs and libraries not distributed with the current component.
– Support for runtime configuration files that allows the custom configuration of components that are installed with statically based runtime managers, for example, the modification of runtime parameters after the package has been installed in the destination path.
– Support for source code distribution of the components.
– Inclusion of metainformation in the description of the components to assist in the preservation of the components.
– A packaging manager that helps during the installation, upgrade, and uninstallation of components that is compatible to existing packaging managers such as RedHat packing manager (RPM).

The packaging toolkit has been tested on the 2.0 beta version of the Globus Toolkit. As part of this test, various packages have been released that are targeted toward different platforms but also include various sets of components. In future versions it is expected that administrators may choose a particular set of components and the platform and may get a custom-assembled package in return. The automatic generation of metadata related to this package is integrated into the Grid Information Infrastructure that is implemented as part of MDS. Thus, it will be possible to query for installations of the Globus Toolkit and the supported features on various domains. The benefit of using a native C implementation of Grid components is based on their speed.

6 Thin Deployment

Although the packaging mechanism for the Globus Toolkit allows software architects to develop software, services, and components that can be installed and configured by system administrators and Grid component designers, it is still quite complex because of the requirement to support sharing, persistence, and updates. To simplify the task of deployment on a client side, many projects suggest developing thin clients that expose the scientific problem-solving process through convenient Web-based interfaces. A good example for such a Grid-based application is the astrophysical computing portal (see Figure 3) [19] allowing

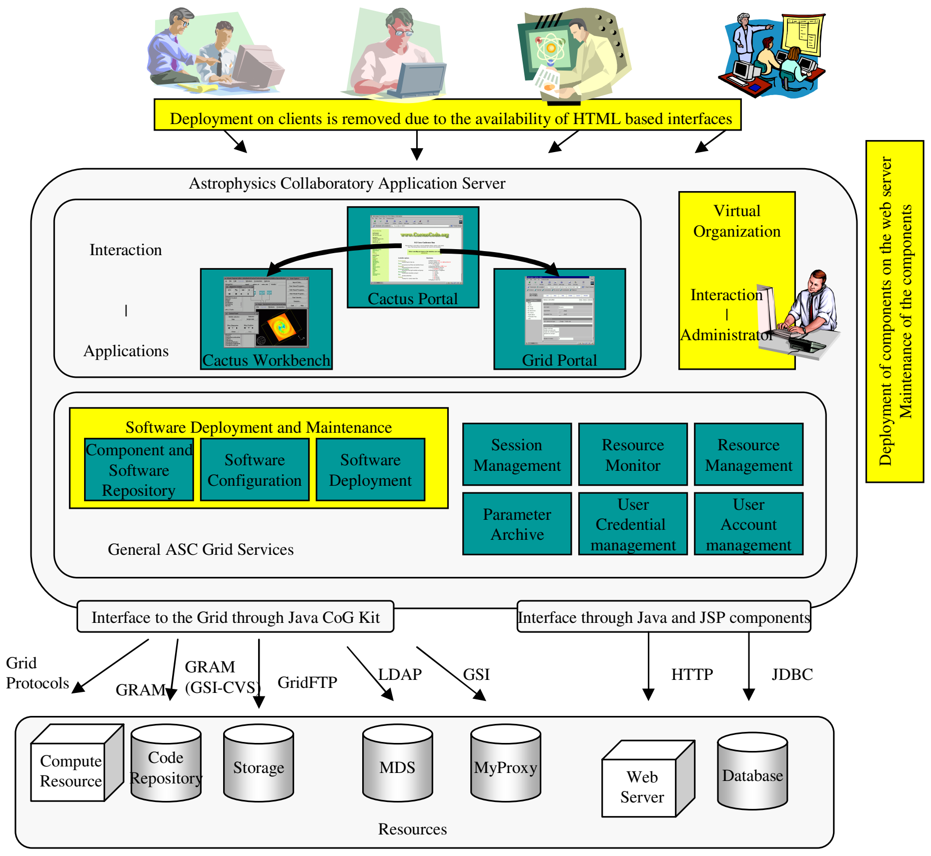

Deployment on clients is removed due to the availability of HTML based interfaces
Astrophysics Collaboratory Application Server
Interaction
|
Applications
Cactus Workbench
Cactus Portal
Grid Portal
Virtual Organization
Interaction
|
Administrator
Deployment of components on the web server
Maintenance of the components
Software Deployment and Maintenance
Component and Software Repository
Software Configuration
Software Deployment
Session Management
Resource Monitor
Resource Management
Parameter Archive
User Credential management
User Account management
General ASC Grid Services
Interface to the Grid through Java CoG Kit
Interface through Java and JSP components
Grid Protocols
GRAM
GRAM (GSI-CVS)
GridFTP
LDAP
GSI
HTTP
JDBC
Compute Resource
Code Repository
Storage
MDS
MyProxy
Web Server
Database
Resources

Fig. 3. Structure of the ASC application server showing the various components.

scientists to access supercomputing resources and interact as required by the astrophysical community.

This application contains two different aspects making it interesting for the deployment of components in computational Grids. First, the thin clients enable application users to easily interact with the system without updating their local software, as all communication is performed through a Web-based portal using HTML and DHTML technologies. If state-of-the-art Web browsers are available on the application users' clients, no additional installation task is required. Naturally, the designer of the software must put special care into fulfilling the application users' needs and must provide sufficient guidance for administrators to install such a portal. Customization for users is provided by saving the state of the last interaction with the portal. Second, the portal is used to develop reusable components as part of the Cactus framework.

The portal provides access to familiar tools such as a GSI-enhanced CVS. Other users are able to access components from a shared component repository.

The Globus Toolkit provides the necessary security infrastructure. Additional administrative overhead is needed to maintain such a portal and to determine policies for access to the software repository, for the software configuration, and for automatic software deployment.

7 Slender Deployment

Practical experience with slender Grid services has shown that the user interface provided by the current generation of browsers is too limited or requires in many cases unreasonably long startup cost. At SC2001 in Denver we demonstrated how the use of slender services and components can support component development and deployment in computational Grids (see Figure 4). Slender clients are developed in an advanced programming language and can be installed through a Web-based portal on the local client. This approach allows the creation of advanced portals and the integration of locally available executables as part of the problem-solving process. Furthermore, it enables the integration of sophisticated collaborative tools provided by a third party.

Although the creation of slender clients can be performed in any programming language, we have chosen in our prototype to use the Java programming language. We rely on the Java CoG Kit [17] for the interface to Grid resources

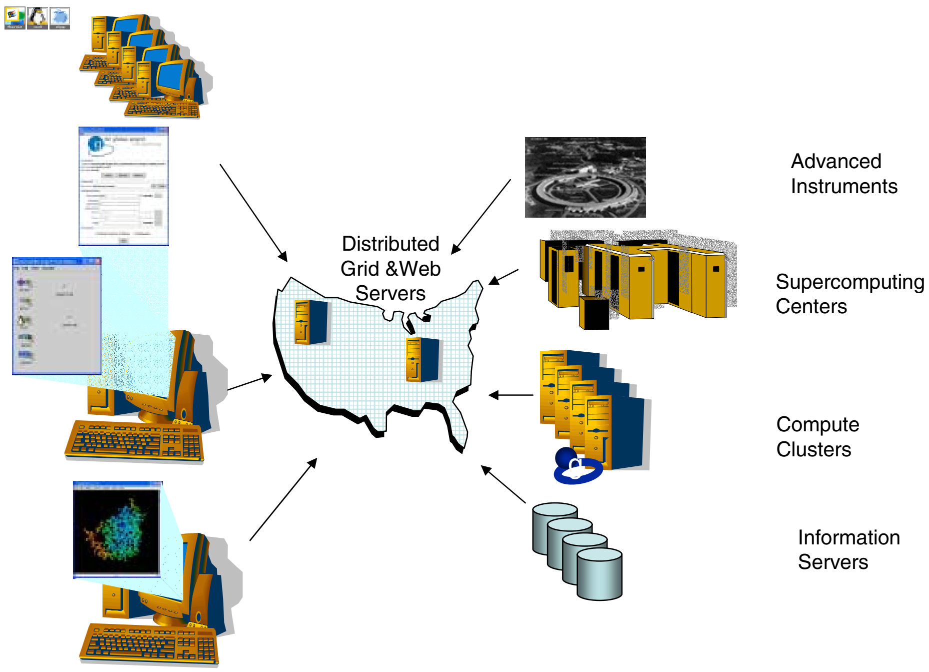


Fig. 4. The slender client environment allowing to significantly simplifying the deployment and maintenance of clients accessing Grid functionality.

that can be accessed through the slender clients. For our demonstration we developed two popularly used components and deployed them through the Java Web Start technology. Java Web Start provides a deployment framework that allows client users with access to a local resource to install Java components that can be accessed from the cache of the local browsers. Thus it reduces the installation time if the component is used more than once. Additionally, it provides an automatic framework for updating the component if a new one is placed on the Web server. Using the protocols defined by the Globus Project, we achieved interoperability between the Globus Toolkit and our Web-enabled slender clients. Tests confirmed the usability of our approach as part of the deployment strategy for Grid computing. Indeed, users not familiar with the Web Start technology were able to deploy the components in just two minutes on their clients. This is in dramatic contrast to other similar Grid software that is sufficiently complex so that users typically must attend a training session to learn about the installation process. Although the packaging toolkit described earlier is aimed to improve this situation, the developers must maintain a significant variety of binary releases that are to be installed with an installshield-like capability on Linux/Unix and Windows platforms. While using Java for this task, we cover a significant portion of the target machines and are even able to support the Macintosh platform, so far not explored by the packaging effort. During SC2001, we also demonstrated the integration of programs natively installed on the client in a Grid components framework. Specifically, we obtained data via a secure connection to a Globus Toolkit-enabled information resource and displayed it with the help of a natively compiled molecular viewer (rasmol). This easy deployment methodology enables the inclusion of new resources and services within the computational Grids. Furthermore, we are able to integrate certificates in components that can be used to guarantee authenticity and portability between deployed components.

8 Comparison of the Scenarios

Experiments with the various deployment scenarios in Grid settings revealed a number of advantages and disadvantages of each approach. In Table 1 we summarize our results. Each of the approaches can access native components in C or FORTRAN, with relatively low overhead. Nevertheless, using a native compiler will have performance advantages while accessing libraries written in C and Fortran. Such advantages, however, come at a price, since no uniformly accepted component framework exists. In contrast, significant benefits can be achieved by using Java as the component integration language [8]. The benefits include implicit integration of documentation and metainformation as part of the Java packaging. These packages can be signed with the standard Java keytool to generate signatures for improving the authenticity of the packages before download. Component repositories can thus be implemented as part of existing Web services, enabling others to share their components with the community easily. Moreover, as pointed out before, the Java Web Start technology provides

Table 1. Comparison between the various scenarios

	Feature	Thick	Slender	Thin
1	Primary interface to C/FORTRAN	native fast	JNI medium	Third tier slow
2	Primary language	C/Fortran, Java	Java	HTML
3	Possible Java programs	application	applets applications	applets
4	Component versioning	libtool	tag	N/A
5	Component interoperability through signature	no	yes	N/A
6	Component metainformation	libtool, rpm	certificates	HTML tag
7	Component repository	Web-server, cvs	Web-server	Gsi-cvs
8	Source repository	Web-server, cvs	Web-server	N/A
9	Portable GUI	Tcl/Tk, Qt	Swing	HTML
10	Sophistication of the GUI	high	high	low
11	Interactivity not limited to browser [20]	yes	yes	no
12	Speed of interface interaction	high	high	low
13	First-time activation cost	high	low	none
14	Cost for subsequent use	low	low	high
15	Support for power users	yes	yes	no
16	Incremental component update	difficult	easy	N/A
17	Integration of client-side programs	yes	yes	no
18	Access restriction to client by sophisticated policies	no	yes	no
19	Standard Component Framework	no	Beans/EJB [16]	none
20	Sandboxing	difficult	yes	no
21	Desktop integration	no	yes	no
22	Offline operation	yes	yes	no
23	Automatic installation of supporting components	no	yes	no
24	Deployment in Grids	distributed	distributed	replicated
25	Deployment protocol	To be defined	JNLP	none

a usable mechanism for installing such signed components on local clients. Other advantages are the availability of a sophisticated user interface development environment that interfaces with the desktop and can so far not be replicated with HTML/XML technologies.

Java does, however, have potential disadvantages. In the slender scenario, portability is defined by the availability of a JVM for the targeted Grid resource. Since some Grid resources lack sufficient support for Java, these resources must be interfaced through native libraries. Another disadvantage may be the restrictions in the address space or the speed numerical calculations are performed (though studies show that the Java performance can be dramatically improved [12]). In our experience many application users and designers are initially pleased with an HTML-based interface but quickly experience frustration because the Interface does not provide for enough flexibility and speed during a continuous period of use. An example of such an application is given by the use of our Java-based MDS browser [18], which has thousands of users (in contrast to its original CGI counterpart, which was simply too slow in continuous use). The ability to sandbox client-side applications is a further advantage of the slender client and enables one to create sophisticated high-throughput compute resources similar in spirit to SETI@home [10], Condor.

9 Related Work in the Grid Community

Work on component assembly and deployment was being performed in Grid-related activities even before the term Grid was coined [16]. More recently, the Globus Project has defined a schema for a metacomputing directory service [6] that allows one to store metainformation about components that are installed on remote resources. Moreover, with the input of the Globus Project team, University of Tennessee researchers have completed a schema proposal to the Grid Forum [11]. Such developments will be carefully watched by new efforts such as the European GridLab project and the DOE Science Grid projects, which will need to address the deployment issue of Grid components and services. The research conducted in this paper will be beneficial for this work. Other relevant efforts are, for example, [7,5]

10 Summary

In this paper we have outlined three scenarios that significantly affect the deployment strategy of components within Grids. Although none of the deployment strategies is all encompassing, together they solve many aspects of component deployment in Grids. We found the strategy of signed slender clients to be significantly superior to the thin-client approach supported by other communities. In fact, we found that often artificial requirements were put in place to prevent developers from considering slender client deployment strategies, even when such strategies allow integrating previously written client software. Additionally, we have shown that with the availability of Java we were able to deploy components with ease on Java-enabled platforms including Solaris, Linux, Windows, and Macintosh. We view our continuing research in this field as essential for the acceptance and the success of Grids.

Acknowledgments

This work was supported by the Mathematical, Information, and Computational Science Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38. DARPA, DOE, and NSF support Globus Project research and development. We thank Ian Foster and Brian Toonen for valuable contributions in this ongoing research. This work would not have been possible without the help of the Globus Project team. Globus Toolkit and Globus Project are trademarks held by the University of Chicago.

References

1. Ncsa in the box packages. http://www.ncsa.uiuc.edu/TechFocus/Deployment/, 2001.
2. The DOE Science Grid, 2001. http://www-itg.lbl.gov/Grid/.
3. The Globus Security Web pages, Dec. 2001. http://www.globus.org/security.
4. Bill Allcock, Eric Blau, and Michael Bletzinger. The Globus Packaging Desing Document, 2001. http://www-unix.globus.org/packaging/rfc.html.
5. D. C. Arnold and J. Dongarra. The netsolve environment: Progressing towards the seamless grid. In *Proc. International Workshop on Parallel Processing*, 2000.
6. K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid Information Services for Distributed Resource Sharing. In *Proc. 10th IEEE International Symposium on High Performance Dis-tributed Computing*, 2001. http://www.globus.org.
7. J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke. Condor-g: A computation management agent for multi-institutional grids. pages 55–66, 2001.
8. Vladimir Getov, Gregor von Laszewski, Michael Philippsen, and Ian Foster. Multi-Paradigm Communications in Java for Grid Computing. *Communications of ACM*, 44(10):119–125, 2001. http://www.globus.org/cog/documentataion/papers/.
9. Steve Graham, Simeon Simeonov, Toufic Boubez, Glen Daniels, Doug Davis, Yuichi Nakamura, and Ryo Neyama. *Building Web Services with Java: Making Sense of XML, SOAP, WSDL and UDDI.* SAMS, December 2001.
10. W. T. Sullivan III, D. Werthimer, S. Bowyer, J. Cobb, D. Gedye, and D. Anderson. A new major SETI project based on Project Serendip data and 100,000 personal computers. In *Proc. of the Fifth Intl. Conf. on Bioastronomy.*, Astronomical and Biochemical Origins and the Search for Life in the Uni-verse., 1997. http://setiathome.ssl.berkeley.edu/woody_paper.html.
11. Jeremy Miller. Grid Software Object Specification, 2001. http://www-unix.mcs.anl.gov/gridforum/gis/reports/software/software.pdf.
12. José E. Moreira, Samuel P. Midkiff, Manish Gupta, Pedro V. Artigas, Peng Wu, and George Almasi. The NINJA Project. *Communications of ACM*, 44(10):102 – 109, 2001.
13. OMG. CORBA: Common Object Request Broker Architecture, 2001. http://www.omg.org.
14. John Towns. The Alliance Virtual Machine Room, 2001. http://archive.ncsa.uiuc.edu/SCD/Alliance/VMR/.
15. Andre van der Hoek, Richard S. Hall, Antonio Carzaniga, Dennis Heimbigner, and Alexander L. Wolf. Software deployment: Extending configuration management support into the field. *Crosstalk, The Journal of Defense Software Engineering*, 11(2), February 1998. http://www.cs.colorado.edu/serl/cm/Papers.html#CROSSTALK98, http://www.stsc.hill.af.mil/crosstalk/1998/feb/deployment.asp.
16. Gregor von Laszewski. An Interactive Parallel Programming Environment applied in atmospheric Science. In N. Kreitz, editor, *Making its Mark, Proceedings of the 6th Workshop of The use of Parallel Processors in Meteorology*, pages 311–325. World Scientific European Centre for Medium Weather Forecast, Reading, UK, 1996.
17. Gregor von Laszewski, Ian Foster, Jarek Gawor, and Peter Lane. A Java Commodity Grid Kit. *Concurrency and Computation: Practice and Experience*, 13(8-9):643–662, 2001. http://www.globus.org/cog/documentation/papers/cog-cpe-final.pdf.

18. Gregor von Laszewski and Jarek Gawor. Copyright of the LDAP Browser/Editor, August 1999.
19. Gregor von Laszewski, Michael Russell, Ian Foster, John Shalf, Gabrielle Allen, Greg Daues, Jason Novotny, and Edward Seidel. Community Software Development with the Astrophysics Simulation Col-laboratory. *Concurency and Computation*, to be published, 2002. http://www.globus.org/cog.