Toward GT3 and OGSI.NET Interoperability: GRAM Support on OGSI.NET[*]

James V.S. Watson, Sang-Min Park, and Marty Humphrey

Department of Computer Science, University of Virginia, Charlottesville, VA 22904, USA

Abstract. OGSI.NET is the implementation of the Open Grid Services Infrastructure (OGSI) that leverages the Microsoft .NET Framework. OGSI.NET and the Globus Toolkit combine to create a comprehensive platform for computational science by supporting the emerging Grid protocols on Windows and Linux/UNIX, respectively. A significant challenge in building OGSI.NET is interoperability with the Globus Toolkit, both in terms of the rendering of individual services (OGSI-compliance and more recently WSRF-compliance) and also conformance to higher-level protocols developed in the Globus project and in the Global Grid Forum. This paper presents the design and experiences of implementing the Globus GRAM protocols on OGSI.NET. A major challenge was to easily and securely create processes as specific target users in the Windows environment. Differences between GT3 GRAM and OGSI.NET GRAM are described and an overview of WSRF.NET GRAM is presented.

1 Introduction

It is generally believed that the foundation of next-generation Grids will be the Open Grid Services Architecture (OGSA)[1], which is an overall vision for Grid computing that combines the strengths of projects such as Globus [2] and Legion [3] with Web Services. OGSA endorses the *service-oriented architecture* as the foundation of Grid Computing. The introduction of the largely abstract OGSA was accompanied by the Open Grid Services Infrastructure (OGSI)[4], which attempted to specify the low-level interfaces and—to a certain extent—the behaviors of the individual services in an OGSA-compliant Grid. OGSI defined a particular rendering of the service by using both standard and non-standard uses of the XML, SOAP, and the Web Services Description Language (WSDL). The Grid community actively contributed to the definition of OGSI through the Global Grid Forum (GGF) standardization process. OGSI.NET [5] is the implementation of the Open Grid Services Infrastructure (OGSI) that leverages the Microsoft .NET Framework. OGSI.NET and the Globus Toolkit combine to create a comprehensive platform for computational science by supporting the emerging Grid protocols on Windows and Linux/UNIX, respectively.

An important use of a computational grid is allowing users to submit jobs to the grid without needing explicit knowledge of which machines are being used or logging

[*] This work is supported in part by the US National Science Foundation under grants ACI-0203960 (Next Generation Software program), SCI-0438263 (NSF Middleware Initiative), the US Department of Energy through an Early Career Grant, and Microsoft Research.

onto each machine manually. For example, a user may desire to run a parameter-space problem in which hundreds of similar jobs can be executed in parallel. The user would prefer to authenticate himself only once to the computational grid, though many machines may be used to execute these parallel jobs. The user will also prefer to create these many job specifications with a minimum of effort and not have to configure the job specifications based on which machines are being utilized for each job.

A significant challenge in building OGSI.NET is interoperability with the Globus Toolkit, both in terms of the rendering of individual services (OGSI-compliance and more recently WSRF-compliance [6]) and also conformance to higher-level protocols developed in the Globus project and in the Global Grid Forum. Specifically with regard to remote execution described above, the challenge is to conform to the Grid Resource Allocation Manager (GRAM) [7] of the Globus Toolkit. This paper reports the implementation experiences of the development of GRAM-supporting services on OGSI.NET. A major challenge was to easily and securely create processes as specific target users in the Windows environment. This work is the *first* attempt to expand the Grid to include remote job execution on Windows machines via GRAM, as GT3 is only able to run on Windows machines now in a limited client-side mode via the Java Cog [8]. A .NET client was also written that runs on Windows and can submit jobs to machines running GT3 or OGSI.NET. These results provide the basis for remote execution currently being created in our implementation of WSRF on .NET [9] [10].

The next section reviews GRAM. Section 3 describes Windows security issues that came up as a result of trying to implement the same functionality of GRAM on Windows machines. Section 4 explains OGSI.NET GRAM, interoperability between Windows and Linux machines, and an example usage of this set of services. Section 5 gives a brief overview of WSRF.NET GRAM. Section 6 concludes this paper.

2 Review of GRAM

GRAM running on GT3 allows a user to easily submit a job and have it execute on the machine running the GRAM set of services. The Globus Resource Specification Language (RSL) [7] is an XML schema-defined language that is used by the user to specify the job submission. Many aspects of a job submission can optionally be included, such as the starting directory, user environment, and non-local input files. Substitution values are also allowed so that values can be better controlled and updated. For example, if the only changes in a parameter space problem were a single argument and related output filename, the user could specify the value like so: <rsl:substitutionDef name= "ArgValue"> <rsl:stringElement value="50"/> </rsl:substitutionDef>. The user would then be able to specify this value in multiple locations in the job submission like so: <rsl:substitutionRef name="ArgValue"/>.

Job submissions are authenticated by the use of a gridmap file located on each server, which contains a mapping between an X509 certificate DN and a local user name. The job submission is signed with the user's X509 certificate. The DN from the signature is then compared to the entries in the gridmap file. If a match is found, the job submission is accepted and the job will be run as the local user specified in the gridmap file. By running the job as a local user, the damage a malicious user can cause is limited to whatever access rights the local user already holds.

3 Windows Security

In this section, we describe the relevant aspects of Windows security that in some cases were leveraged and in some cases had to be overcome in order to implement GRAM in OGSI.NET. Implementing the GRAM set of services on Windows proved to be rather challenging because of the security mechanisms that are available in Linux and are not easily accessible in Windows. The major hurdle in Windows security is creating a new process (in this case, the actual job to run on the machine) as a different user than the user asking for the process to be created.

Security in Windows is based, in part, on passing user tokens to authentication methods. The well-documented methods available to create a new user token require a user name and password combination in clear text. If these well-documented methods are used, there are a number of possibilities for implementation. In one case, the user securely enters in the password for each job submission. Obviously, this is undesirable because it does not allow a user to sign-on once and for the grid services to authenticate the user on each machine. Another option is to include the user name and password pair in the gridmap file. This option has two major problems. If the file containing the passwords is read by a malicious user, that user can access the system as any user that is listed in the file. By mapping the DN to a local user name, the ability to read the gridmap file does not make it easier for a malicious user to gain access to the system by impersonating another user because it is assumed that the DN is hard to forge. Another problem is that the gridmap file will need to be changed every time the user changes his password on the machine, which would likely result in an updating nightmare for the system administrator.

UNIX and Linux systems allow a setuid bit to be set on files. When this bit is set on an executable, the executable may change the effective user id so that it can access resources that are inaccessible to the user that ran the executable. The call to change the effective user id only requires that the setuid bit be set on the executable.

There is a method on Windows called NtCreateToken() that is located in ntdll.dll. In order to use the NtCreateToken() method, the user running the executable that calls NtCreateToken() must be given the Create Token privilege. The call to NtCreateToken() is not as simple as the call to setuid() in *nix systems, which only requires the new user id. NtCreateToken() requires 12 input parameters that must be properly set for a new user token to be created. Since this method is undocumented in Windows help files, our code was influenced by other code that required the use of NtCreateToken(): CVSNT, Cygwin, and GUI-Based RunAsEx. Currently, the user token information also must be located on the machine where this code is running. This means that a user name cannot be intended to refer to an account name that is on a domain other than the local machine.

An important difference between the setuid bit and the NtCreateToken() method is the extent of the privileges. On *nix systems, the ability to impersonate other users is confined to the executable that has the setuid bit set, but any user with the proper permissions can run that executable. On Windows systems, the privilege to call

NtCreateToken() is given to users, instead of executables. This means that the user who runs any executable that eventually calls NtCreateToken() must have the Create Token privilege. This greatly increases the amount of code that is now susceptible to malicious users who would then gain the ability to run an executable as any user.

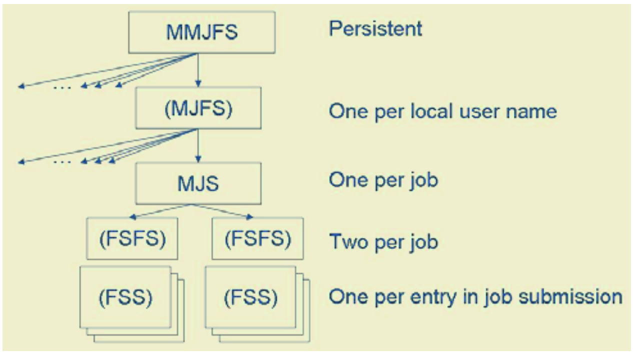

Fig. 1. OGSI.NET GRAM Architecture

4 OGSI.NET GRAM

The service architecture of OGSI.NET GRAM is based on GT3 GRAM and appears in Figure 1. The top service is the Master Managed Job Factory Service (MMJFS). This persistent service waits for new job submissions. When a job submission is received and the request is authenticated with the gridmap file, a Managed Job Factory Service (MJFS) may be created. In order to efficiently use local resources, at most one MJFS for each local user name that appears in the gridmap file will be running at any time. When the first job to run as a given local user name is submitted, a new MJFS is created and the job submission is forwarded to the new MJFS. Otherwise, the MMJFS will forward the job submission to the MJFS that already exists for the given local user name. When a MJFS receives a job submission, it creates a new Managed Job Service (MJS) that will actually execute and monitor the job as it runs locally on the system. The MJS creates two File Stream Factory Services (FSFS), one corresponding to standard output and one for standard error. Each FSFS will then create as many File Stream Services (FSS) as are specified in the job submission. The reason that a user can specify multiple locations is to make it easier for job results to be distributed. For example, a single user submits a job, but the user may wish all members of his group to receive the results. The RSL allows the user to not have to worry about sending the results out after the job has completed. The MJFS, FSFS, and FSS names appear in parentheses because the services are used by the system to satisfy the job submission, but it is not necessary for the user submitting the job to know that they exist. When a user submits a job, a reference to the newly created MJS is returned.

4.1 Differences Between OGSI.NET and GT3 GRAM

The GRAM set of services in GT3 also includes a Resource Information Provider Service (RIPS) that actually accesses the underlying operating system to get system-specific information, such as the scheduling and file systems. The MMJFS and MJS instances will subscribe to the RIPS to acquire information about the local system and job state changes, respectively. While we understand that this type of information is arguably necessary for OGSI.NET MMJFS and MJS instances, we do not believe that it is necessary to conform to the specific interfaces of the Globus RIPS at this time.

When the requested executable is actually run, the executable is called by invoking the Windows method CreateProcessAsUser() and passing in the user token created by NtCreateToken(). Except for the specified executable, everything in the OGSI.NET GRAM set of services run as the user with the Create Token privilege. The GRAM set of services in GT3, however, create each MJFS as the specified local user. This reduces the amount of security vulnerabilities that may allow a malicious user to gain access to the system as any user.

4.2 Interoperability

In many ways, the single criteria for success for OGSI.NET remote job execution is to faciliates clients on both Windows and Linux machines to send job submissions to both Windows and Linux machines with a minimum of user interaction.

By solely using GRAM in GT3, it is possible to take a Linux client and submit a job to a Linux machine running GT3. The Globus Alliance offers a Java CoG Kit [8] which offers some client functionality, but does not offer a client that can use the version of GRAM in GT3. Java CoG Kit version 1.1 only supports up to version 2.4.0 of the Globus Toolkit. Analysis of the client code offered in the GT3 source download revealed that it is possible to invoke a Java-only client that can use GRAM in GT3, thus allowing a Windows client to also submit jobs to a Linux machine running GT3.

The job submission sent from the client must be signed with the user's certificate in order for the server to properly authenticate the job submission request. In GT3, the versions of the WS-Security specification and other Web Services specifications are provided as part of the GT3 implementation, while OGSI.NET uses Microsoft's Web Services Enhancements 2.0 (WSE 2.0) [11]. In early development, we had difficulty getting our implementation of WS-Security to interoperable with the version of WS-Security in GT3. However, this has since been resolved as both implementations have recently conformed to the Basic Security Profile of the Web Services Interoperability group (WS-I [12]).

4.3 Example Usage

The prototypical use case is shown in Figure 2. At the time of this writing, due to incompatible security protocols inherited from tooling on both the Windows and Linux platforms, currently, there is no client that can run on Linux that can submit a job to OGSI.NET GRAM (although we expect this to change shortly). GT3 already provides a Linux client that can submit a job to GT3, so the arrow in Figure 2 indicating that functionality is not applicable to the work described in this paper. We have our client run from a Windows Server 2003 machine to show that the Windows client and server do not have to run the exact same operating system (i.e., Windows XP).

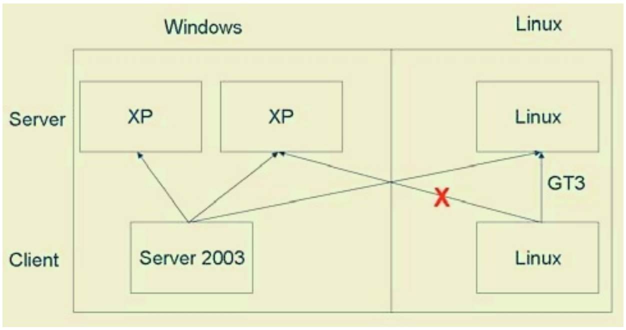

Fig. 2. Example Machine Configurations

The Windows client that we developed invokes either an OGSI.NET client or a Java GT3 client (via the Java Cog), depending on the machine that is being contacted. Other than the simple switch to know which client code to invoke, the arguments are exactly the same.

While GridFTP has been implemented as part of the OGSI.NET project, it is not currently fully-integrated into the OGSI.NET hosting environment such that a server can move files on behalf of a user automatically. Therefore, the client also examines the job submission file and downloads the first standard output and standard error files to the client's current running directory. This allows a client to easily view the results of running the job without having to manually check the output files on the server. This GridFTP functionality requires "grid-proxy-init" and "globus-url-copy" from the Java CoG Kit and GridFTP instances running on the servers.

In our testing, to ensure that the job was being run as the proper local user we set up the same file on both OGSI.NET servers that explicitly denied access to user Foo1. The same X509 certificate DN referred to different local user names on the two OGSI.NET servers (users Foo1 and Foo2). When the job submission included the DOS command "type" and specified the above file, one server displayed the contents of the file, while the other server denied the user access to that file and printed the error to standard error.

Running this client exposed the fact that Windows has a concept similar to real and effective user ids like *nix systems. For example, executing a .NET program as the job submission that displays the value System.Environment.UserName will show the name of the user that has the Create Token privilege. Running the Cygwin "whoami" command instead will show the local user name that is in the gridmap file.

5 WSRF.NET GRAM

In this section, we introduce a new GRAM implementation currently being developed on top of WSRF.NET. The Web Service Resource Framework (WSRF), announced in

January 2004, is an effort to create stateful web services by defining “WS-Resources” [6]. It provides an abstraction of stateful resources which could be identified by existing web service technologies (e.g., WS-Addressing). Globus Tookit version 4 (GT4) is now being developed based on the WSRF concepts. GRAM in GT4 has different features from GT3 GRAM. The Job (process) is now modeled as a WS-Resource, and a client can interact with a job through operations defined in WSRF portTypes and GRAM Services. In other words, a client can submit a job through an operation in the GRAM Factory Service, and query the job’s state using one of the resource property operations defined in WSRF.

WSRF.NET is an implementation of full set of WSRF and WS-Notification specifications on Microsoft .NET [9]. WSRF.NET GRAM is an implementation of GRAM services using WSRF.NET libraries and tooling. GT4 GRAM defines two WSRF-compliant web services, ManagedJobFactoryService (MJFS) and ManagedJobService (MJS). It also defines an xml-based job description language. WSRF.NET GRAM exposes these two web services and accepts the job description. The ManagedJobFactoryService keeps information of hardware and software resources (e.g., localResourceManager) as WS-Resources and reflects their state as resource property document. WS-Resources in the ManagedJobService are information of process.

When a job-creation operation is called by a client, MJFS creates a new resource as a Windows process and the process is executed under the local user name which is different from the user name running ASP.NET infrastructure. The process handle is used to expose a process. WSRF.NET has a mechanism to save and retrieve the process handle to/from a database so that the process information can be sustained. The client may query MJS about the resource property of process (e.g., process state) using operations defined in the WSRF Resource Property portTypes. When the job’s state changes, MJS notifies the client of new state using the WS-Notification libraries implemented in WSRF.NET. After receiving the ‘done’ or ‘failed’ notification, the client terminates.

The Globus Toolkit has been using the proxy certificate for WS-Security, and it has been the biggest challenge for interoperability. In WSRF.NET we support WS-Security processing with GT proxy certificate, thus GT4 GRAM and WSRF.NET GRAM are interoperable. In other words, users can submit the job to WSRF.NET GRAM services using GT4 client tools. In our future release, we are planning to provide a secure and reliable authorization mechanism, a delegation service, and file movement service all of which will be incorporated in WSRF.NET GRAM.

6 Conclusion

Remote execution is a fundamental operation for Grids. Prior to the work described in this paper, there was no Grid-standards-compliant mechanism for remote execution on the Windows platform. In this paper, we described the design and implementation of GRAM for OGSI.NET, focusing on the security model and issues that had to be overcome. We also described our prototype implement in WSRF.NET. We are currently building on this support to create the University of Virginia Campus Grid (UVaCG), using WSRF.NET and GT4 as the foundation of the Grid. By combining this new support for remote execution with our continuing support for remote data access via GridFTP and higher-level abstractions and GUIs, we have taken a significant step toward the realization of the Grid as a ubiquitous platform for computational science.

References

[1] Foster, C. Kesselman, J. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Draft of 6/22/02. http://www.gridforum.org/ogsi-wg/drafts/ogsa_draft2.9_2002-06-22.pdf
[2] Globus Project. http://www.globus.org
[3] A.S. Grimshaw, A.J. Ferrari, F.C. Knabe and M.A. Humphrey, "Wide-Area Computing: Resource Sharing on a Large Scale," IEEE Computer, 32(5): 29-37, May 1999.
[4] S. Tuecke et. al. Open Grid Services Infrastructure (OGSI) Version 1.0. Global Grid Forum. GFD-R-P.15. Version as of June 27, 2003.
[5] G. Wasson, N. Beekwilder, M. Morgan, and M. Humphrey. OGSI.NET: OGSI-compliance on the .NET Framework. In *4th IEEE/ACM International Symposium on Cluster Computing and the Grid (ccGrid 2004)*. Chicago, Illinois. April 19-22, 2004
[6] K. Czajkowski., Ferguson, D., Foster, I., Frey, J., Graham, S., Sedukhin, I., Snelling, D., Tuecke, S., Vambenepe, W. 2004. The WS-Resource Framework. http://www-106.ibm.com/developerworks/library/ws-resource/ws-wsrf.pdf
[7] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, S. Tuecke. A Resource Management Architecture for Metacomputing Systems. *Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing*, pg. 62-82, 1998.
[8] G. von Laszewski, I. Foster, J. Gawor, P. Lane. A Java Commodity Grid Toolkit. *Concurrency: Practice and Experience*, 13, 2001.
[9] M. Humphrey, G. Wasson, M. Morgan, and N. Beekwilder. An Early Evaluation of WSRF and WS-Notification via WSRF.NET. *2004 Grid Computing Workshop (associated with Supercomputing 2004).* Nov 8 2004, Pittsburgh, PA.
[10] Web Services Resource Framework on the .NET Framework. http://www.ws-rf.net
[11] Microsoft. Web Services Enhancements (WSE). http://msdn.microsoft.com/webservices/building/wse/default.aspx
[12] Web Services Interoperability Organization (WS-I). http://www.ws-i.org