This report was prepared for the

SEI Administrative Agent
ESC/XPK
5 Eglin Street
Hanscom AFB, MA 01731-2100

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

This work is sponsored by the U.S. Department of Defense. The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.



This work was created in the performance of Federal Government Contract Number FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

For information about purchasing paper copies of SEI reports, please visit the publications portion of our Web site (http://www.sei.cmu.edu/publications/pubweb.html).

# Table of Contents

# List of Figures

# List of Tables

# Abstract

Many current technology approaches exist for building systems that have interoperability requirements. This report investigates Open Grid Services Architecture (OGSA), one of the many technologies for accomplishing interoperability, using the T-Check technique. A T-Check is a simple and cost-efficient way to understand what a technology can and cannot do in a specific context. This report describes a T-Check exploration of the feasibility of using OGSA in the context of data management, finding that OGSA (a) provides data storage and retrieval where the specific implementation of the data store implementation is transparent and (b) allows addition or removal of data stores at runtime without affecting system operation. This report is part one of a two-part investigation; part two will look at OGSA in the context of load distribution.

# 1 Introduction

The Integration of Software-Intensive Systems (ISIS) team at the Carnegie Mellon® Software Engineering Institute (SEI) is examining technologies and approaches for the construction of systems that are required to interoperate with other systems, with the purpose of identifying gaps between what these technologies and approaches offer and what users expect of them. The end goal is to provide users with information about what can be expected from the current state of technology and to provide technology suppliers with information about user expectations.

There are many technologies for building systems that have interoperability requirements. Each approach has particular advantages and disadvantages with respect to interoperability, and each works well in some circumstances but not in others [Lewis 04]. In this report, we investigate Open Grid Services Architecture (OGSA), one of many technologies for accomplishing interoperability [Foster 04].

Computing power in terms of processing power, storage capability, and bandwidth has continuously increased over the past two decades. However, computing needs have become even more demanding, creating challenges such as:

- Many complex applications today require substantially more computing power and resources than are provided by traditional computing systems. For example, there is a need for systems capable of processing and storing information in the range of tens of petabytes ($10^7$ gigabytes) [Childers 06].

  Even if it is technically feasible for an organization to acquire the necessary computing infrastructure, it may not make economic sense for a single organization to invest in an expensive but highly capable computing infrastructure, unless a close-to-full resource utilization is justified. For example, a bioinformatics organization running a simulation might require an infrastructure for only 15 days a month. The expensive computing environment therefore remains idle half of the time, wasting resources.
- Heterogeneous computer systems and applications often need to interact and interoperate with each other.

  For example, a cancer research agency needs to distribute, store, and retrieve high-resolution, cancer-related medical images from/to over 30 different medical centers and hospitals [HP 05]. Each hospital or medical center could have systems running on different hardware platforms and software built using different technologies.
- Changing business drivers, needs, and environment pose the biggest challenges, perhaps, to today's enterprises.

  For example, business processes and software systems must rapidly incorporate changes in the marketplace, accounting policies, or laws—not only to meet mandates but also to remain competitive.

® Carnegie Mellon is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

Grid computing, as implemented through OGSA, possibly provides a solution to some of these challenges by allowing collaboration and resource sharing between organizations [Foster 04].

- Grid computing is based on the creation of virtual organizations (VOs) that allow sharing of resources between organizations (see Section 2.1). Typically, an organization can use the resources it owns and controls. However, in a VO, computing resources from various organizations are pooled together, allowing them to utilize resources that are not directly under their control. The pooling of resources leads to better computing capability for all participating organizations without each investing in and maintaining an entire infrastructure.
- Grid systems tend to be heterogeneous and distributed—encompassing a variety of hosting environments (e.g., J2EE and .NET), operating systems (e.g., Unix, Linux, Windows, and embedded systems), devices (e.g., computers, instruments, sensors, storage systems, databases, and networks), and services. Various vendors can provide all of those environments, systems, devices, and services. Grid computing architectures, such as OGSA, assume that resources in a Grid system will be heterogeneous. Virtualization of resources on a Grid using open protocols and standards enables interoperability between heterogeneous elements. For example, underlying computing nodes that are based on different computer architectures and run on different operating systems can contribute CPU cycles in a Grid system.
- A Grid computing architecture promotes loose coupling, decentralization, and service-orientation, enabling rapid incorporation of changes. For example, a storage service provider can use Grid technology to provide technology-neutral data services to its customers on the Internet.

Initially, in the mid-1990s, Grid computing was restricted to the scientific research community [Foster 01]. However, over the past few years, Grid-based applications and infrastructure have been explored in domains such as financial risk analysis, drug discovery, biomedical research, and healthcare [OGF 07].

A T-Check investigation is a simple and cost-efficient way to understand what a technology can and cannot do in a specific context [Lewis 05a]. The goal of this report is to use the T-Check approach to explore the feasibility of using OGSA in the context of data management. Specifically, this T-Check investigation focuses on understanding how OGSA deals with (a) heterogeneous data stores and (b) a need to meet storage demands dynamically. This report is part one of a two-part investigation; part two will look at OGSA in the context of load distribution.

In Section 2, we provide fundamental Grid computing concepts. In Section 3, we define the T-Check elements for the exploration of OGSA. Section 4 provides the details about design and implementation. Finally, in Sections 5 and 6, we discuss our findings and recommendations.

# 2 Grid Computing

Foster provides this three-point checklist for defining a Grid system [Foster 02a]:

1. coordinates resources that are not subject to centralized control
2. uses standard, open, general-purpose protocols and interfaces
3. delivers nontrivial qualities of service

To understand this definition, consider a Grid system that comprises several organizations, as shown in Figure 1. In this example, Organizations A and B provide a cluster of highly capable workstations and/or mainframes that will be used to provide compute cycles to consumers of the Grid system, such as Organization C. Organizations B and D contribute data storage resources.

The workstations, mainframes, and databases are resources on the Grid system. No single organization owns them all, or even all of any resource. Each organization controls its resources by defining its own policies and mechanisms. The sharing organization keeps control over its resources when shared on the Grid. Consumers utilize these shared resources in a coordinated and controlled fashion without having any direct control over them.

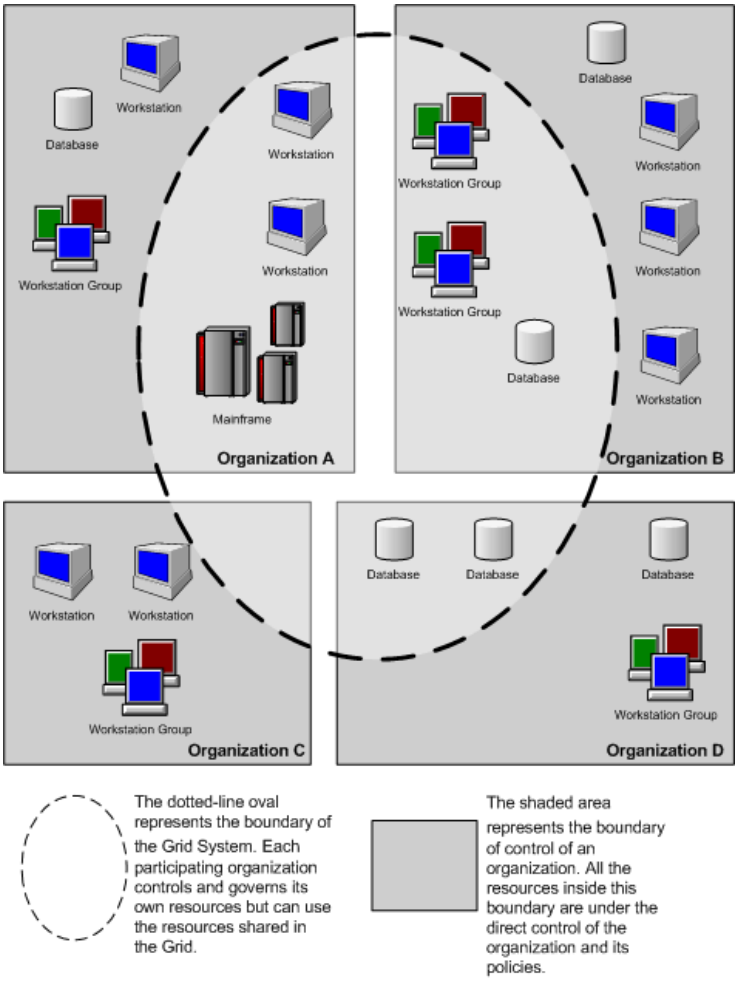


*Figure 1: Example of a Grid System*

A Grid system allows heterogeneous resources to interact, coordinate, and interoperate with each other. This coordination and sharing is possible only if the protocols and interfaces used by all participating organizations are standardized and open. These protocols provide mechanisms such as authentication, authorization, resource discovery, scheduling, and resource access in a Grid system [Childers 06].

## 2.1 VIRTUAL ORGANIZATION (VO)

Understanding the concept of the VO is essential and fundamental for understanding Grid computing and architectures that facilitate the implementation of Grid systems (see Section 2.2). VOs can be viewed as runtime subsets of a Grid system that enable disparate groups of organizations or individuals to share resources in a controlled fashion, so that member organizations can collaborate to achieve a shared goal. A Grid system provides benefit when the quality of service (QoS) it delivers to any participating organization is significantly better than the individual organization can afford to provide by itself [Foster 02a]. Therefore, an organization will see benefit from participating in a Grid system if it cannot (due to technical or economic reasons) create the necessary infrastructure by itself. In many cases, the participating organizations do not have any prior relationships [Foster 01]. Figure 2 shows a Grid system and the creation of three virtual organizations where there are four real organizations contributing resources.

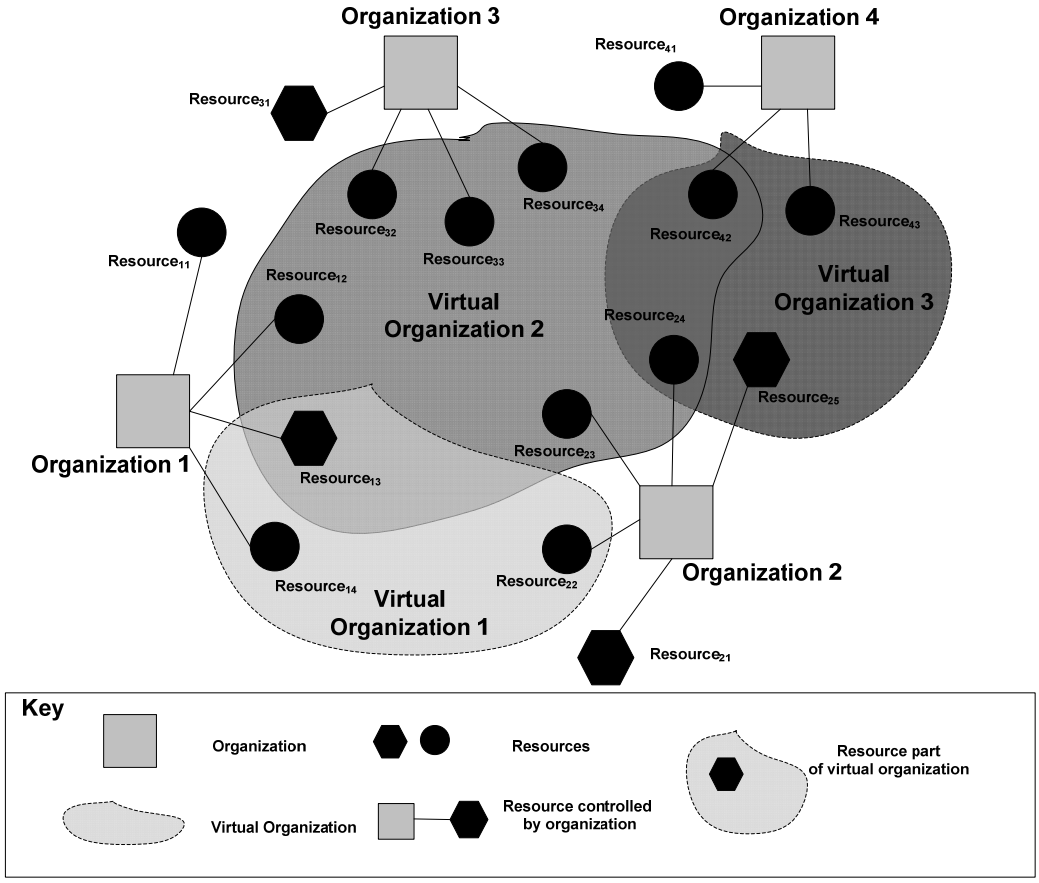


*Figure 2: Virtual Organizations*

There are some important aspects of resource sharing in VOs [Childers 06, Foster 01, Foster 02a]:

- *Sharing is not limited to information exchange and can involve direct access to remote distributed resources such as software, computers, data, databases, and sensors.* For example, a cluster of computers can be used to run a large risk analysis simulation of "what-if" scenarios. In this case, the computing cycles from various computers shared on the Grid are used for the simulation. Coordinated sharing of these computing cycles in a distributed environment is nontrivial compared to simple data or file exchanges.
- *Sharing is conditional.* Each resource owner makes resources available, subject to its constraints and policies. For example, a storage service provider (SSP) might give higher priority to its paying consumers than its nonpaying ones. The SSP may also provide paying consumers a higher quota of storage space or create a policy in which it does not allow data originating from a specific geographical region to be stored on its devices. These polices and mechanisms can be modified during the operation of a Grid system.
- *Relationships in a VO are dynamic and vary over time in terms of the resources involved, the nature of the access permitted, and the participants to whom access is granted.* An organization sharing a resource can decide to drop membership of the VO at any time, or the same organization can decide to share a new resource. Therefore, a VO should provide mechanisms for discovering and characterizing the nature of the relationships that exist within elements of a VO. For example, the number of organizations providing disk storage space can change

over time, resulting in an increase or decrease in the actual storage capacity provided by the VO. The VO and the supporting infrastructure must provide mechanisms to deal with these dynamic changes. In this example, the VO should have a mechanism for identifying the currently registered storage service providers and other parameters such as maximum allowable storage capacity.

- *A shared resource can be used in more than one way, depending upon the context.* For example, a computer's CPU can be used to provide compute cycles and, at the same time, host software that provides infrastructure capability to the Grid system.
- *A shared resource can have membership in more than one VO at the same time.* For example, a shared workstation can provide compute cycles in one VO and storage space on its physical devices for another VO. For example, in Figure 2 resources $Resource_{42}$ and $Resource_{24}$ have membership in VOs 2 and 3.

## 2.2 GRID ARCHITECTURE

Grid architecture defines the elements that are required for establishing and maintaining VOs. It defines basic components—along with their purposes and functions—and the interactions between them [Foster 01]. A layered Grid architecture is presented in Figure 3.

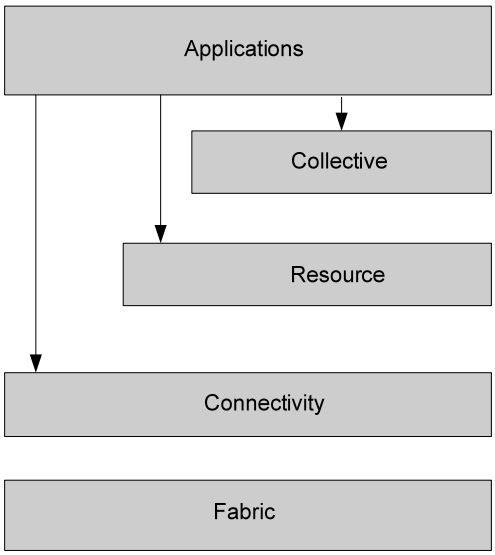


*Figure 3: The Layered Grid Architecture*

Table 1 provides the description of the various layers illustrated in Figure 3.

*Table 1: Layers of Grid Architecture*

| Layer | Description |
|---|---|
| Fabric | This lowest layer in the Grid architecture contains the resources that are shared among VOs using the Grid infrastructure. Some examples of these resources are computational platforms, storage devices, and network resources. These resources may also be logical entities such as a distributed file system or a distributed computer cluster. This layer is not concerned with the internal implementation details of the logical entity. For example, the network protocols used in a distributed file system are encapsulated from the fabric layer of the Grid architecture. Components in the Fabric layer implement the resource-specific operations that are local to the resources. |
| Connectivity | The Connectivity layer is responsible for defining the core communication and authentication protocols required for Grid-specific transactions. Examples of such protocols are TCP/IP, HTTP, HTTPS, and DNS. These communication protocols enable exchange of data between Fabric layer resources and the authentication protocols that are necessary to verify the identity of resources and users in a Grid system.[1] |
| Resource | The Resource layer contains services, APIs, software development kits (SDKs), and protocols for managing resources individually. These management activities include secure negotiation, initiation, monitoring, control, accounting, and payment of sharing operations on individual resources. Two primary classes of Resource layer protocols are<br>1. Information protocols used to obtain information about the structure and state of a resource (e.g., how much free space is available for storage on a data server)<br>2. Management protocols that provide partial control over the actual resource and are used to negotiate access to shared resources (They ensure that requested protocol operations are consistent with the policies under which the resource is being shared. For example, requirements such as QoS and advanced reservation can be specified when they share a resource.) |
| Collective | The Collective layer deals with services, APIs, SDKs, and protocols for managing multiple resources; it contrasts with the Resource layer where the focus is on one specific resource. This layer implements a wide variety of sharing behaviors among a collection of resources without placing new requirements on the individual resources that are shared. Examples of components in this layer are directory services that allow VO participants to discover resources and their properties and *data replication services* that support storage access and management. |
| Application | This layer refers to the actual user applications[2] that run and operate within a Grid system. As shown in Figure 3, it is not mandatory for applications to access only the Collective layer that is directly below it. Applications can also use the Resources and Connectivity layers directly. The ability to bypass the Collective layer provides more flexibility to the application layer. |

[1] Acronyms and initialisms used in this report are defined in Appendix B.

[2] In this technical note, such applications are referred as Grid-based applications.

## 2.3 OPEN GRID SERVICES ARCHITECTURE (OGSA)

OGSA is a standard and open architecture for Grid systems. OGSA is based on fundamental concepts and technologies from Grid computing and Web Services [Foster 02b, Lewis 06]. The primary goal of OGSA is to identify and standardize in a Grid system most of the commonly found services, such as security, job management, resource management, and data management.

OGSA defines a core set of standard service interfaces with their associated semantics for purposes such as state management, fault management, and service creation and management. Both interfaces and semantics are required to build interoperable and reusable services. This common and standard service semantics and interface for a service is called *Grid Service.* A Grid Service is a Web service that adheres to the OGSA standards. OGSA uses the standard Web services interface definition language (WSDL) to define services for creating, naming, managing, monitoring, grouping, and exchanging information among Grid Services [Foster 04, Zhang 05].

## 2.4 GLOBUS TOOLKIT (GTK)

The open source Globus Toolkit 4 (GTK4) contains services, programming libraries, and development tools designed for building Grid-based applications [Foster 06]. It was developed by the Globus Alliance and other contributors from around the world [GlobusAlliance 06]. The main idea behind the Globus toolkit is to provide a fundamental and robust infrastructure, tools, and libraries for creating a Grid system. The toolkit also provides services commonly used by Grid-based applications.

The GTK4 architecture contains three key components [Foster 06]:

1. A set of infrastructure service implementations such as execution management, data access and movement, replica management, monitoring and discovery, and credential management (Most of these services are Web services written in Java.)
2. Three containers that can be used to host user-developed Grid services written in Java, Python, and C
3. A set of client libraries that are used to interact with and invoke GTK4 services as well as user-developed Grid services

These Globus toolkit components fall into five main categories: (1) security, (2) data management, (3) execution management, (4) information services, and (5) common runtime. Figure 4 shows the details of each of these five categories [Foster 06].

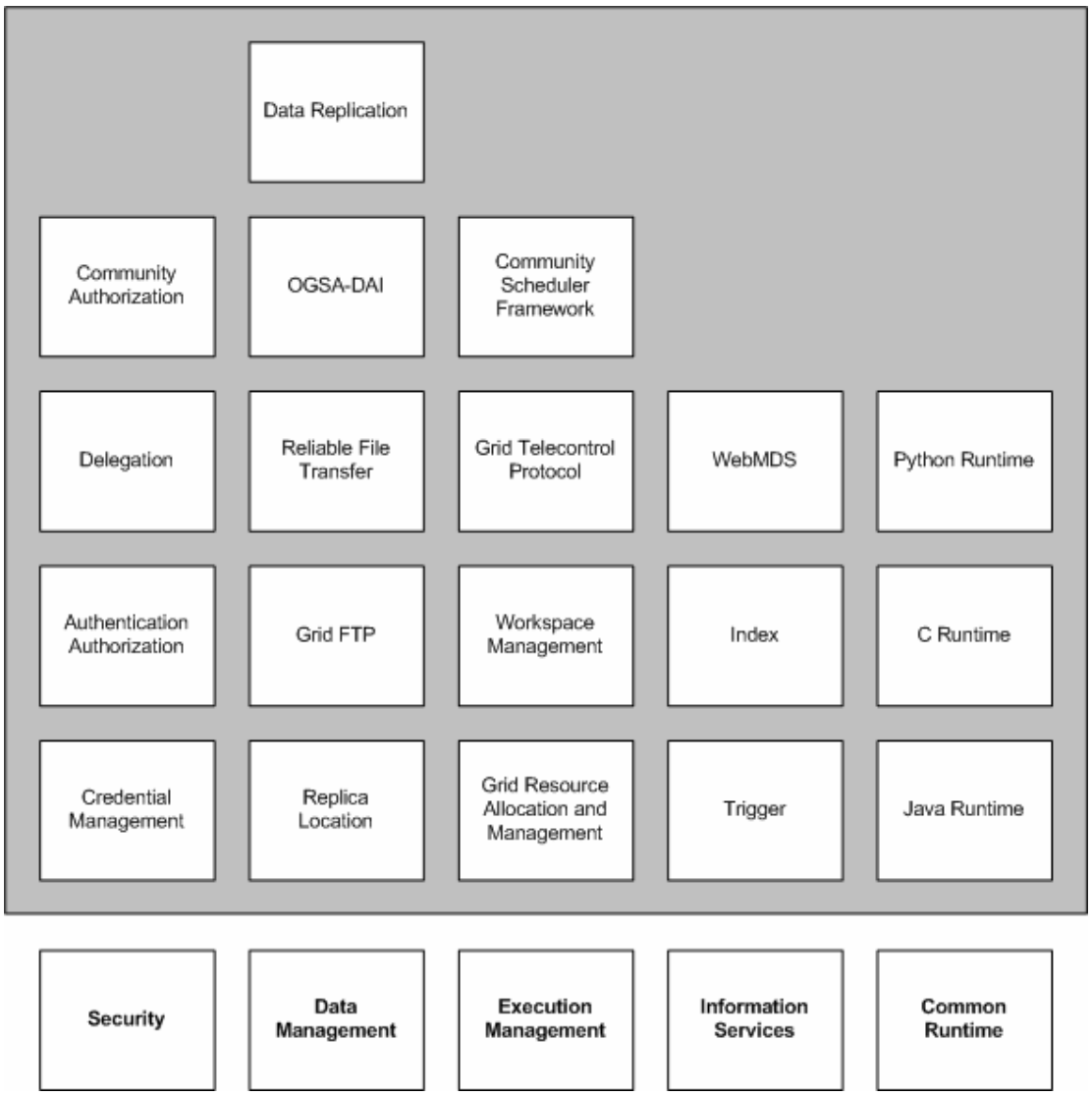


*Figure 4: Primary Components of Globus Toolkit 4*

# 3 Using the T-Check[SM] Approach

The T-Check[3] approach is a technique for evaluating technologies. This approach involves (1) formulating hypotheses about the technology and (2) examining these hypotheses against specific criteria through hands-on experimentation. The outcome of this two-stage approach is that the hypotheses are either sustained or refuted. The T-Check approach has the advantage of producing very efficient and representative experiments that not only evaluate technologies within the context of their future use but also generate hands-on competence with the technologies [Wallnau 01]. A graphical representation of the T-Check process is shown in Figure 5.

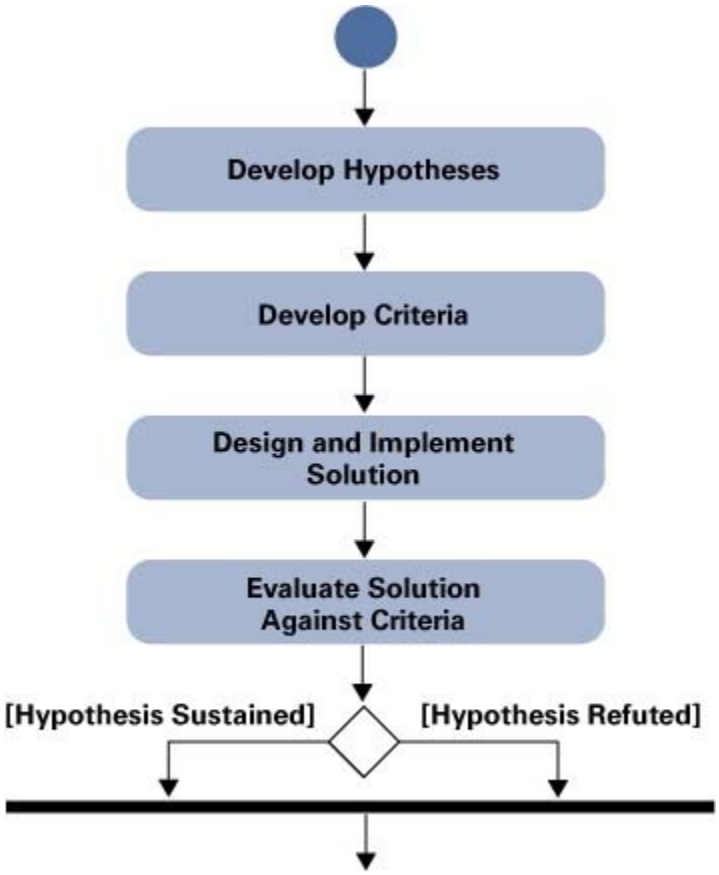


*Figure 5: The T-Check Process for Technology Evaluation*

The T-Check process is part of a larger process for context-based technology evaluation. In this larger process, the context for the T-Check is established, and the expectations from the technology are captured [Lewis 05a].

[3] The T-Check approach was called the model problem approach and is referred to as such in previous SEI technical notes and reports.

### 3.1 T-CHECK CONTEXT

The context for this T-Check investigation is data management in a distributed, data-centric environment without centralized control. Consider the generic scenario shown in Figure 6 where a large volume of data is generated from a continuous data source. This generated data (or a relevant part of it) needs to be stored and retrieved to be processed in some way. Finally, the processed information will be used by data consumers.

This scenario is common in industry and the DoD. In addition, it is not uncommon for data generators, data processors, data stores, and data consumers to be heterogeneous, independent entities. The absence of centralized control is one important distinction between Grid systems and traditional distributed systems. In a distributed system, even though the elements are distributed, they are in most cases under the centralized control of one entity or a small number of entities. Another characteristic of this scenario is the ability to support dynamism, meaning that the amount of data to be stored varies with time.

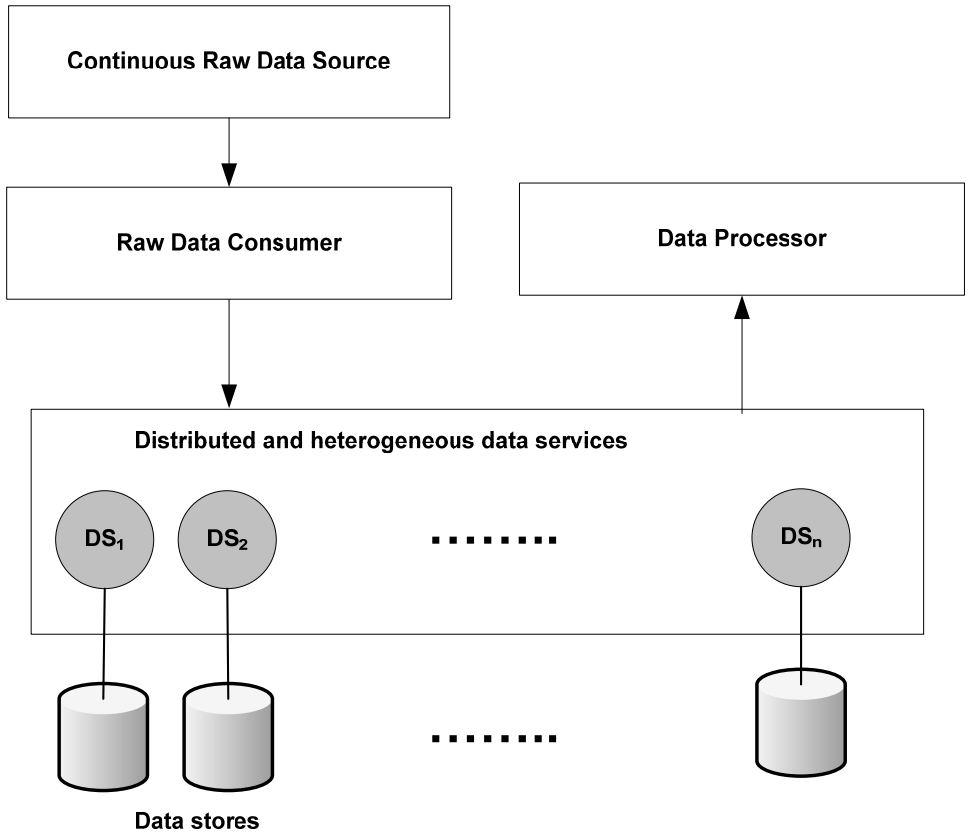


*Figure 6: Context Diagram of a Generic Data Management Scenario*

Table 2 provides detail about the important elements in the scenario illustrated by Figure 6 as they were viewed for this T-Check investigation.

*Table 2: Elements of a Generic Data Management Scenario*

| Element | Description |
|---|---|
| Continuous Raw Data Source | Generates data at regular intervals, for example<br>• weather satellite<br>• traffic flow data sensor<br>• stock quote ticker system<br>• wireless sensor network installed on a bridge<br>• news stream of an online newspaper on the Internet |
| Raw Data Consumer | Fetches raw data from the continuous raw data source and uses the data service to store it in the data stores. |
| Data Services | Capture and store data from a continuous data source<br>The data generated by the data source is dynamic; the data service must have the capability to store all the data transparently—by switching data repositories as soon as a certain storage capacity threshold is reached on a particular database instance, for example. |
| Data Stores | Store all raw data |
| Data Processor | Retrieves data from the data stores and then performs some processing on it |

## 3.2 EVALUATION HYPOTHESES FOR THIS T-CHECK INVESTIGATION

Hypotheses are claims about the technology that will be supported or refuted after the successful completion of the T-Check investigation. For this T-Check investigation, the following hypotheses were defined:

1. OGSA provides data storage and retrieval where the specific implementation or location of the data sources is transparent for raw data consumers and data processors.
2. OGSA allows developers to add new data stores (databases) and remove existing data stores at runtime without affecting the overall operation of the system.

### 3.3 EVALUATION CRITERIA

After the hypotheses are formed, the next step in the T-Check process is to define evaluation criteria for each hypothesis. The criteria associated with the hypotheses in this T-Check investigation are shown in Table 3.

*Table 3: Evaluation Criteria for the T-Check Investigation*

| Hypothesis | Evaluation Criteria |
|---|---|
| OGSA provides data storage and retrieval where the specific implementation or location of the data sources is transparent for raw data consumers and data processors. | 1. A data consumer application accesses a data service by using a unique address and does not need to know the actual location or type of data sources serviced by the data service. For example, the consumer of the data service should not have any dependency on the actual type and location of the database(s). If an Oracle database is replaced with a MySQL database without changing the logical address[4] of the data service, the data consumer application should run without the need to make any code changes.<br>2. The data is distributed and stored in a round-robin fashion in the available databases. For example, if there are three available databases, data records are distributed among them continuously and equally. |
| OGSA allows administrators to add new data stores (databases) and remove existing data stores at runtime without affecting the overall operation of the system. | 1. The data service has an Oracle and a MySQL database. A new MySQL database is added without affecting the overall operation of the system.<br>2. The second MySQL database is removed without affecting the overall operation of the system.[5] |

[4] The logical address of a data service is a unique URL that is used to locate and obtain a reference to the service.

[5] A graceful degradation of the system is assumed when any database is removed. For example, the data on the MySQL database will not be available to the processing service once the MySQL database is removed from the control of the data service.

# 4 Designing and Implementing the Solution

## 4.1 DEFINING A SYSTEM ARCHITECTURE BASED ON THE T-CHECK CONTEXT

The first step in the design process was to create a notional architecture of the system based on the T-Check context discussed in Section 3.1. The goal of creating an architecture was to determine the software requirements for the development environment and runtime environment that was required for implementing the T-Check. Figure 7 illustrates the system architecture designed for this T-Check investigation; the elements of the architecture are described throughout the rest of this section.

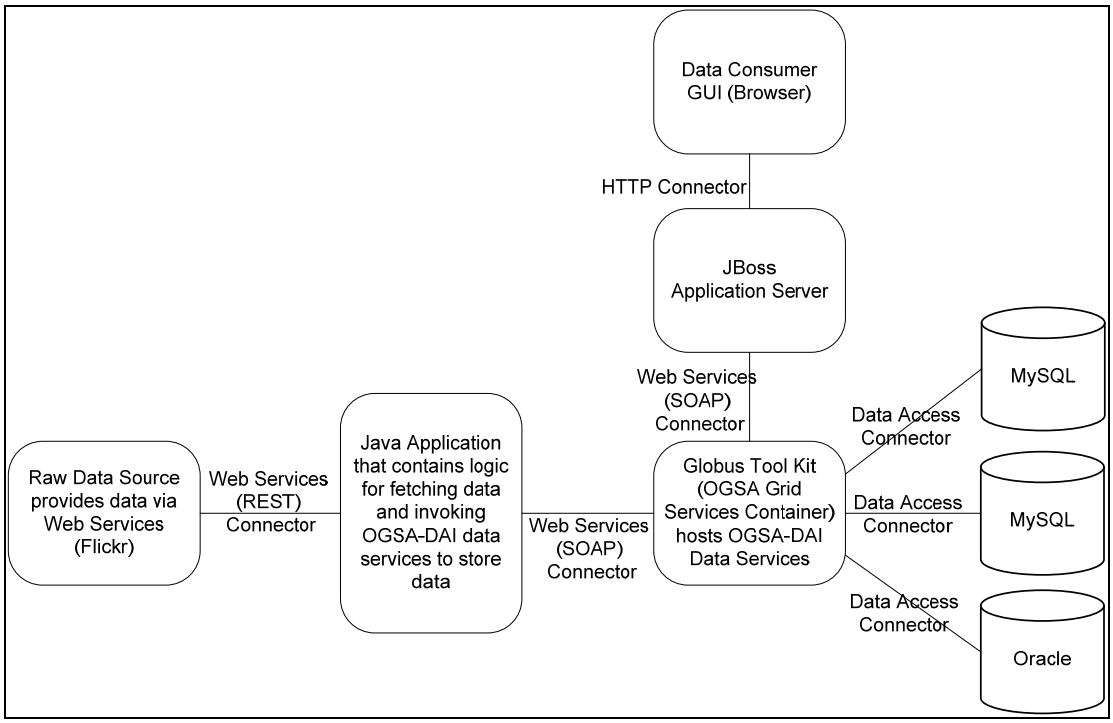


*Figure 7: Architecture for the T-Check Solution*

## 4.2 SELECTING A WEB SERVICE AS A SOURCE OF RAW DATA

Given the notional architecture and the scenario for the T-Check examination (Section 3.1), we evaluated various online Web services as potential raw data sources. The important requirement on these Web services was that they should provide a continuous[6] stream of raw data using a standard Web service interface. We evaluated the following three Web services:

1. A weather Web service that provides temperature information based on date and geographical coordinates [NWS 07]

[6] A continuous stream of raw data in the T-Check context means a data source that provides a significantly large number of raw data records. For this T-Check investigation, any source that provided more than 200,000 records was sufficient.

Although this Web service provided a simple WSDL interface, it did not produce a sufficient volume of data and did not provide information for many calendar dates that we tested.

2. Google's search Web Services Application Programming Interface (API). The Google API imposed a restriction of 1000 searches in one calendar day, a limitation that meant that it was not a good match for the T-Check investigation. In addition, creating the data model for storing the resulting data would have required a large amount of effort due to its complexity, which would have defeated the simplicity characteristic of the T-Check approach.
3. Flickr, an online photo management and sharing application [Flickr 07] provides an API that can be used to write applications that access and use public data such as photos, tags, and profiles [FlickrAPI 07]. This service provides many popular request-response formats such as SOAP, representational state transfer (REST), and XML-RPC, along with API kits in various programming languages such as Java, .NET, Perl, and PHP. We decided that this service was well suited for the needs of the T-Check investigation for the following reasons:
    a. The online application provides a public Web services API that can be used free of cost for noncommercial applications.
    b. The semantics and data model of the raw data are relatively easy to understand, which reduces the learning curve of the API and allows more focus on the actual technologies being evaluated.
    c. The API provides a wide selection of operations, such as getting the profile information of subscribers of the Web site, geographical locations of places where the images were taken and the brand of camera they were taken with.
    d. Bulk data can be obtained from the online application making it a continuous data source, one of the key requirements of this T-Check investigation.
    e. The API supports various request-response formats and provides a toolkit for Java, which was our choice of implementation language.

### 4.3 INSTALLING AND CONFIGURING THE GLOBUS TOOLKIT 4 (GTK4)

The next step was to create a Grid environment. The complete version of GTK4 was installed on a Linux machine. Extensive documentation in the form of a quick start guide and a detailed administrator's guide is provided on the Globus Web site [GlobusAlliance 06]. The installation of GTK4 was straightforward; it required simply following the instructions in the online documentation. A simple example Grid service called MathService was deployed to the Grid container to verify that the installation was done correctly [Childers 06].

### 4.4 DISCOVERY OF OGSA-DATA ACCESS AND INTEGRATION

With the Globus toolkit installed and configured, our next step was to investigate the data access support provided by OGSA and the toolkit. At the time of defining the T-Check context and notional architecture, we knew little about the various data access capabilities provided by GTK4. We discovered OGSA-Data Access and Integration (OGSA-DAI), an open source middleware component that supports data access and integration in a Grid system using Grid services [OGSA-DAI 07a]. OGSA-DAI provides capabilities for querying, updating, transforming, and transferring data using Web services in a data-resource-independent way. Basic lower level OGSA-DAI Web

services can be combined to create complex higher-level services for more complex data-intensive operations that are capable of performing business-specific data operations. For example, a simple OGSA-DAI data operation is to insert a record into a database table. A higher-level data operation is to insert a new customer record, which internally could be performing multiple simple insert operations into multiple tables.

After an initial literature review, we decided to use OGSA-DAI to implement the data access and retrieval functionality for the following reasons:

- OGSA-DAI offers data integration services that can be deployed within a Grid system [OGSA-DAI 07a].
- OGSA-DAI provides support for various types of data stores, including relational databases, XML files, and plain text files. It also supports most of the commonly used relational databases—including Oracle and MySQL, the two relational databases we planned to use.

Figure 8 shows the OGSA-DAI architecture and Table 4 provides a description of its elements. OGSA-DAI has a document-oriented interface implemented using Web services conforming to the Web Services Interoperability (WS-I) basic profile and the Web Services Resource Framework (WSRF) [OGSA-DAI 07c, WS-I 06, WSRF 06]. The WSRF implementation of OGSA-DAI is compatible with GTK4 and enables stateful Web services—an important OGSA requirement [Childers 06, Myer 04]. The WSRF-compliant version of OGSA-DAI was used in this T-Check investigation because it is compatible with the Globus Toolkit's implementation of WSRF.

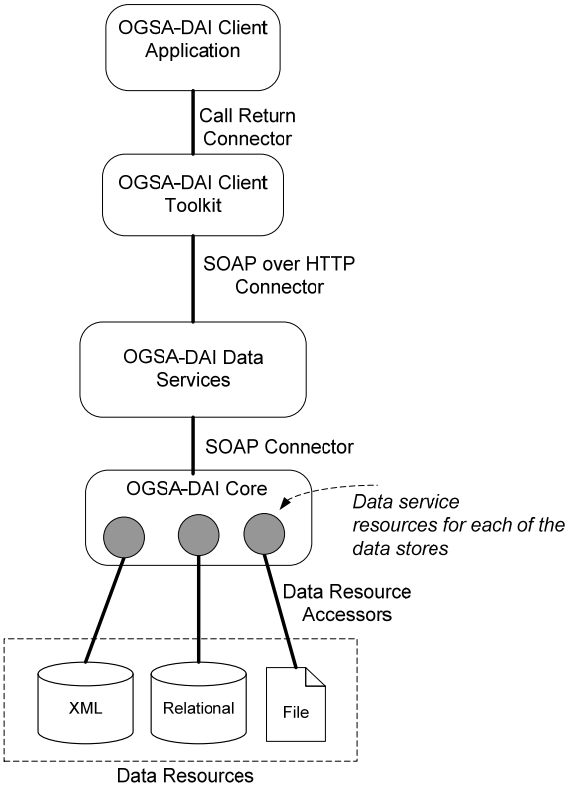


*Figure 8: OGSA-DAI Architecture*

*Table 4: Elements of OGSA-DAI Architecture*

| Element | Description | Responsibilities |
|---|---|---|
| OGSA-DAI Client Application | OGSA-DAI clients can use the OGSA-DAI client toolkit to access any data service. | Uses the OGSA-DAI data services to access the data resources, which not only simplifies the development process but also allows the client to access both WSRF and WS-I data services transparently |
| OGSA-DAI Client Toolkit | This toolkit is a set of higher level APIs that encapsulate some of the complexity of accessing data. | • Provides mechanisms to construct, send, and receive data requests and responses.<br>• Acts as a wrapper that isolates the specific implementation of data services in either WSRF or WS-I |
| OGSA-DAI Data Service | This component is responsible for providing a Web services interface. | Provides a Web services interface to the data service resources that reside inside OGSA-DAI core |
| OGSA-DAI Core | This component is the container for all data service resources. | Receives requests from the data services and passes them to the corresponding data service resource |
| Data Service Resource | This component represents the actual data resource. Each data service resource has a corresponding data resource that it accesses using a data resource accessor. | Is responsible for execution of perform[7] documents, generation of response documents, data source access, and session management |
| Data Resource Accessors | The data resource accessor is the connector between a data service resource and the corresponding data resource. There is a one-to-one mapping between data resource accessors and data resources. | Connects data service resources with their data resources<br>(Currently, OGSA-DAI supports data resource accessors for XML files, relational databases, and file systems. Custom data resource accessors for new data resource types can be implemented.) |
| Data Resources | These are the physical data stores that are exposed using OGSA-DAI services. Three types of data stores are currently supported by OGSA-DAI: relational databases, XML databases, and file systems. | Store data passed from data resource accessors |

[7] A perform document describes the actions that a data service resource should take on behalf of the client. Each action is known as an *activity*. OGSA-DAI includes a large number of activities for performing common operations such as database queries, data transformations, and data delivery.

### 4.5 BUILDING AND DEPLOYING OGSA-DAI WSRF

Having installed GTK4, we then installed OGSA-DAI WSRF using the following steps:

1. Verify the installations of Java 1.4.x, Apache Ant 1.5/1.6 [Apache 07], Globus Toolkit 4.0.x Java Web service core, Oracle database, and MySQL database because these are all prerequisites for installing OGSA-DAI.
2. Complete a source build of the OGSA-DAI sources to create a binary distribution. The source build was straightforward on Linux and Windows XP. We had to modify the standard build file to make it compatible with Java version 1.4 because we had installed JDK 1.5 on our machines and the source and target builds needed to be version 1.4. Once we identified this problem, it was a simple change in the standard build file.
3. Deploy the OGSA-DAI WSRF binaries to a Web services container. In our case, the container was the embedded Apache Axis Web services container included in the GTK4 distribution. OGSA-DAI provides both a command-line as well as a GUI-based interface for deploying the binaries into the container. We used OGSA-DAI command line scripts written in Apache's Ant for deployment.
4. Verify that the actual deployment took place without errors. This deployment was first done on a Windows machine. Although the deployment script never showed any errors, the actual deployment was unsuccessful because we were never able to run any OGSA-DAI services in a Windows XP environment. After a few unsuccessful attempts in the Windows environment, we decided to test the installation of the OGSA-DAI in a Linux environment. We considered the Linux installation successful after testing a simple OGSA-DAI service that was provided with the installation. This example was tested by invoking the OGSA-DAI service from the same Linux machine. We did not pursue the verification of the deployment on the Windows machine because it was not critical to our evaluation.

### 4.6 DEPLOYING AND EXPOSING NEW DATA SERVICES ON OGSA-DAI

Once OGSA-DAI WSRF was deployed into the Grid container,[8] we added data services and exposed them to external clients, using the three-step process summarized in Table 5. While command-line and GUI-based mechanisms are available for performing the steps, we chose to use the command-line version. All the services were deployed without problems.

[8] The Grid container is a runtime environment similar to J2EE-compliant servers that is provided with the Globus Toolkit. The container provides the necessary infrastructure to host Grid services.

*Table 5: Summary of the Process for Deploying Data Services and Data Service Resources*

| Step Description | Step Details |
|---|---|
| 1. Deploy OGSA-DAI data service(s) in the Web services container that runs inside the Globus Toolkit | a) Obtain a unique name for the data service. (Note: a unique name is provided.)<br>b) Configure parameters for<br>• service configuration (dynamic or not dynamic)<br>• the maximum number of concurrent requests that each data service resource can support<br>• the maximum length of the queue that stores requests when the maximum number of concurrent requests limit is reached |
| 2. Deploy OGSA-DAI data service resource(s)<br>(We deployed four data services resources, as shown in Figure 9.) | a) Create a properties file with database characteristics for each data service resource (see Appendix A for an example).<br>b) Install Java Database Connectivity (JDBC) drivers for MySQL and Oracle by placing them in the "drivers" folder before invoking the deployment script for the data service resource. |
| 3. Expose data service resource(s) | a) Map each OGSA-DAI service to its data service resources.<br>b) Expose data service resources. They can be exposed as ordinary or dynamic.<br>• Ordinary exposure of data service resources requires a restarting of the GTK container.<br>• For dynamic exposure, the service should be marked configurable when it is first deployed. |

## 4.7 RUNTIME VIEW OF THE T-CHECK SOLUTION

We decided to deploy four OGSA-DAI data services as shown in Figure 9, which shows the detailed runtime architectural view of the solution implemented for this T-Check investigation. We mapped three services to a MySQL database and one to an Oracle database. All four dynamic services were deployed into a GTK4 container running on a Linux machine. OGSA-DAI dynamic services are configurable at runtime and can be deployed and removed from the GTK4 container without restarting the container.

Together with Figure 9, Table 6 and Figure 10 provide a complete explanation of the runtime view for this T-Check solution. Table 6 on page 21 provides details about each architectural element shown in Figure 9. Figure 10 on page 24 shows the interaction between components.

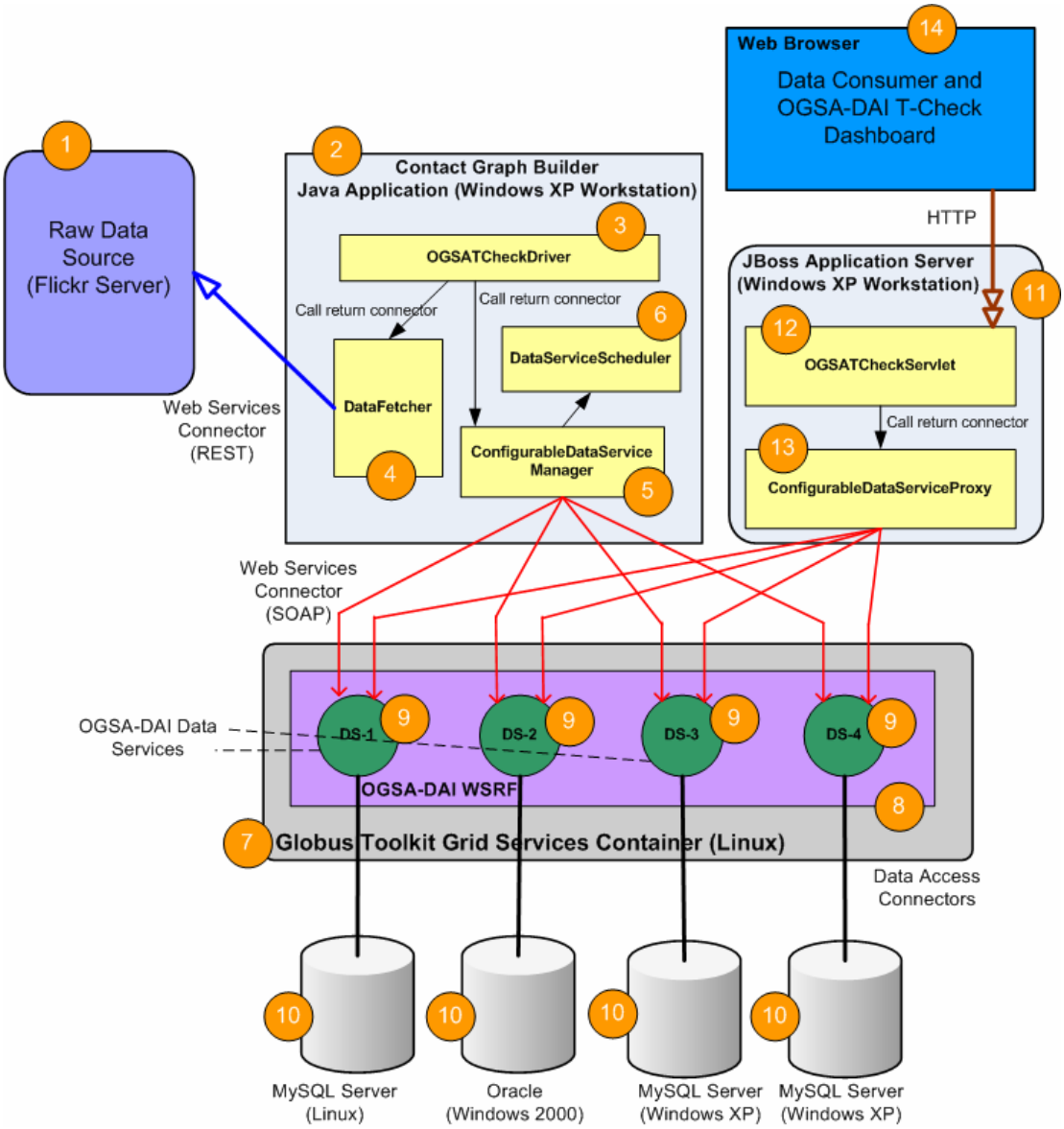


*Figure 9: Detailed Runtime View of the T-Check Solution*

*Table 6: Architectural Elements and Their Responsibilities*

| Number Shown in Figure 9 | Element | Description | Responsibilities |
|---|---|---|---|
| 1 | Raw Data Source (Flickr Server) | External component that is an online photo management and sharing application<br><br>(Authentication is required before any data can be accessed from this service [see section 5.1.3].) | Provides the raw contact information as a Web Services (REST) interface [Fielding 00] |
| 2 | Contact Graph Builder | • Java application running on a Windows XP workstation<br>• Contains OGSATCheckDriver, DataFetcher, ConfigurableDataServiceManager, and DataServiceScheduler | Creates a contact graph for any given subscriber |
| 3 | OGSATCheckDriver | The entry point of the Java application | • Coordinates control flow between the DataFetcher component and the ConfigurableDataServiceManager component using simple call-return connectors<br>• Gets the raw data from the DataFetcher and provides it to the ConfigurableDataServiceManager |
| 4 | DataFetcher | Invoked by the OGSATCheckDriver to fetch raw data from the raw data source<br><br>(The DataFetcher is executed in the same Java Virtual Machine [JVM] space as the OGSATCheckDriver.) | • Performs the authentication logic to connect to the Flickr server before invoking any Web services<br>• Invokes the Web service to fetch the raw data from the server (The fetched data is passed to the OGSATCheckDriver.) |

*Table 6: Architectural Elements and their Responsibilities (cont.)*

| Number Shown in Figure 9 | Element | Description | Responsibilities |
|---|---|---|---|
| 5 | ConfigurableDataServiceManager | Invoked by the OGSATCheckDriver and executed in the same JVM space as the OGSATCheckDriver to provide access to the OGSA-DAI data services | • Initializes the OGSA-DAI data services<br>• Connects and invokes the OGSA-DAI data services using SOAP[9]<br>• Provides business-specific[10] data retrieval and storage methods<br>• Fragments the data and asks the DataServiceScheduler to get the next available OGSA-DAI data service |
| 6 | DataServiceScheduler | Uses round-robin mechanism to provide next available OGSA-DAI data service to the ConfigurableDataServiceManager | • Provides next active OGSA-DAI data service instance to the ConfigurableDataServiceManager for storing and retrieving data<br>• Keeps track of active and inactive OGSA-DAI services |
| 7 | Globus Toolkit Grid Services Container | Runtime environment for the OGSA-DAI WSRF and OGSA-DAI data services | Provides a runtime environment for hosting and executing WSRF Web services (see Section 2.4). |
| 8 | OGSA-DAI WSRF | Framework that makes OGSA-DAI data services available | Supports OGSA-DAI data services at runtime with a set of Java libraries<br><br>(OGSA-DAI data services cannot be deployed directly to the GKT4 runtime container. They require OGSA-DAI WSRF at runtime for their execution [see Section 4.5].) |
| 9 | OGSA-DAI Data Services | Configurable data services deployed on the GTK4 container on top of the OGSA-DAI WSRF (see Section 4.6); can be accessed by the ConfigurableDataServiceManager and the DataServiceProxy | Provide a SOAP interface to the underlying relational databases<br><br>(This interface is used by other components, the ConfigurableDataServiceProxy and ConfigurableDataServiceManager, that want to use the database.) |

[9] OGSA-DAI provides a Java-based client toolkit that encapsulates the logic of invoking the OGSA-DAI data services by providing Java APIs instead of using SOAP.

[10] Two examples of business-specific data access methods are to (1) check if a particular user has already been added to the contact graph and (2) store the actual contact information.

*Table 6: Architectural Elements and their Responsibilities (cont.)*

| Number Shown in Figure 9 | Element | Description | Responsibilities |
|---|---|---|---|
| 10 | Relational Database Servers | Relational database servers corresponding to the OGSA-DAI data services<br><br>(In this T-Check investigation, Oracle and MySQL databases were used.) | Retain the information provided by the OGSA-DAI data services |
| 11 | JBoss Application Server | A runtime container | Hosts the OGSATCheckServlet and ConfigurableDataServiceProxy |
| 12 | OGSATCheck Servlet | Java Servlet | • Invokes the ConfigurableDataService-Proxy to get the required data from the databases<br>• Creates the output Hypertext Markup Language (HTML) that is displayed |
| 13 | ConfigurableDataServiceProxy | Used by the OGSATCheck Servlet to obtain data from the databases | • Initializes and connects to the OGSA-DAI data services<br>• Provides business-specific data retrieval methods[11]<br>• Invokes the "deploy" operation and removes configurable data services dynamically<br>• Keeps track of the OGSA-DAI data services that are alive at any point |
| 14 | Data Consumer and OGSA-DAI T-Check Dashboard | HTML-based graphical user interface | Provides the following functionality<br>• Ability to monitor the status of each OGSA-DAI data service<br>• Ability to change the status of any OGSA-DAI service (active or inactive)<br>• Ability to show the data stored in each relational database |

[11] Unlike ConfigurableDataServiceManager, ConfigurableDataServiceProxy does not provide methods for storing information.

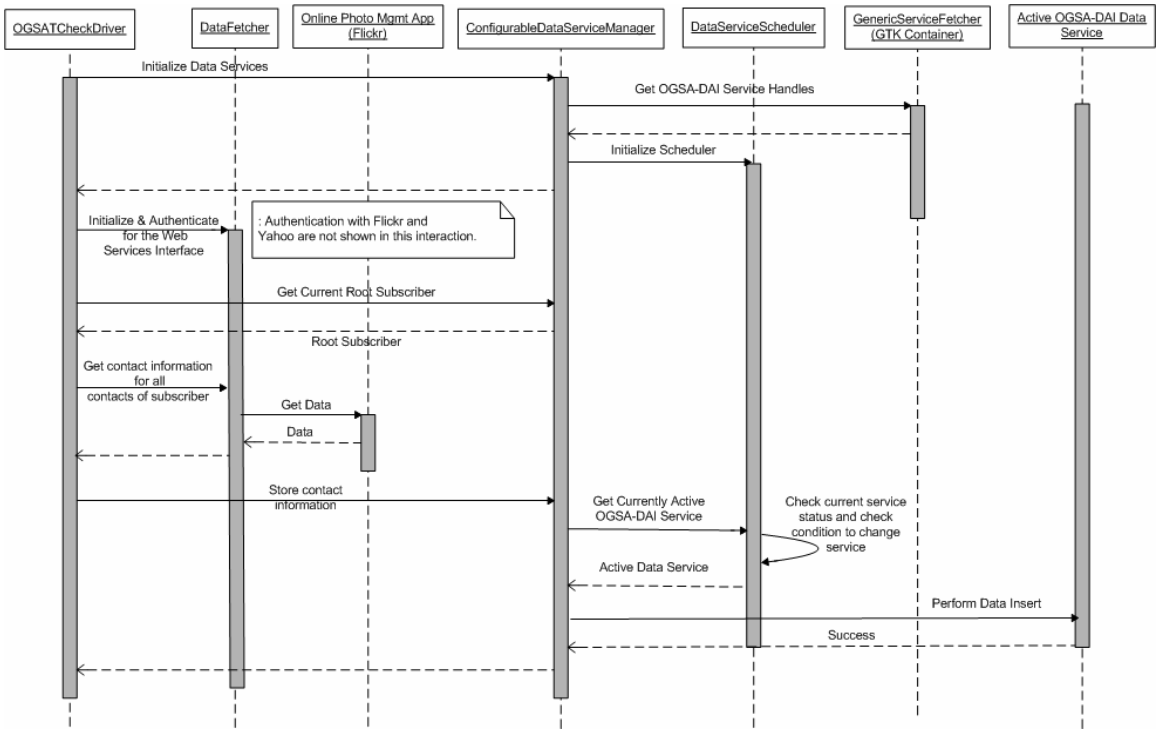


*Figure 10: Sequence Diagram Showing the End-to-End Interaction between Various Components*

## 4.8 USING WEB SERVICES TO ACCESS DATA FROM EXTERNAL WEB SITE

With the OGSA-DAI data services deployed, our next step was to implement the Web services that obtain raw data from the online photo sharing and management application (see Section 4.2). For the scope of this T-Check investigation, we decided to get the profile information of each subscriber to the online photo management Web site. The raw data collected was the publicly available contact information for the application's subscribers. Each subscriber on the Web site has one or more contacts. Each of these contacts is also a subscriber of the Web site and has its own set of contacts.

The goal of the Contact Graph Builder application (see number 2 in Figure 9 and Table 6) is to create a contact graph for any given subscriber, as shown in Figure 11. We call this subscriber the Root Subscriber ($RS_1$ in Figure 11). Getting raw data by invoking the Web services was easy once authentication was performed. The authentication mechanism used by the Web site was confusing, though, and there was not enough documentation available to facilitate usage. We spent a considerable amount of time investigating the authentication mechanism. The details of our experience are explained in Section 5.1.2. The DataFetcher subcomponent (see number 4 in Figure 9 and Table 6) has the responsibility of authenticating and fetching the data using the Web services. An open source Java toolkit provided by the online photo management Web site was used to invoke the Web services based on the REST protocol.

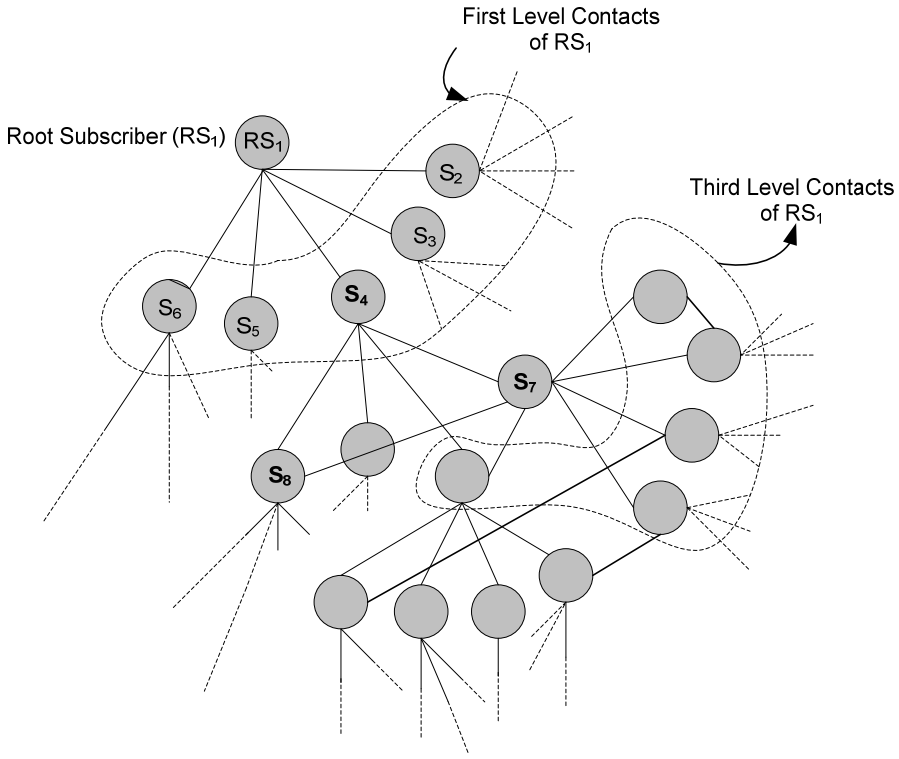


*Figure 11: Contact Graph Showing the Model of the Raw Data*

## 4.9 CREATING A ROUND-ROBIN SCHEDULER FOR OGSA-DAI DATA SERVICES

As shown in Figure 9 on page 20, we deployed four OGSA-DAI data services. All of these data services were logical replicas, meaning they had the same underlying database schema and supported the same operations. One of our goals was to test the dynamic deployment and removal of OGSA-DAI data services. In support of that goal, we decided to implement the DataServiceScheduler, a round-robin scheduler (number 6 in Figure 9 and Table 6) that was responsible for alternating between all of the available data services. The DataServiceScheduler switches between services when one of the following conditions occurs:

- The currently active service has become inactive for some reason.
- The number of records to be stored in a particular database has reached the chunk size. The chunk size can be configured at the time of starting the Contact Graph Builder application.

As explained before, we decided to deploy and use four OGSA-DAI data services. However, one of these services is considered a "sacred service," meaning that is always alive. This configuration is necessary because its underlying database stores metadata[12] required for building the contact graph.

[12] The metadata is information about the current root subscriber and a list of all the contacts who have been root subscribers at some previous point of the graph creation.

## 4.10 INTEGRATING DATA ACCESS AND STORAGE BY USING OGSA-DAI SERVICES

After successfully performing the authentication and invoking the Web service, we wrote logic for creating the contact graph that considers the following:

- As mentioned before, one of our requirements is a continuous data source. We achieved this by looping over all possible unique contacts for a subscriber and their contacts recursively. For example, in Figure 11 we first fetched and stored data for all the first-level contacts of $RS_1$ ($S_2$, $S_3$, $S_4$, $S_5$, and $S_6$). After this, $S_2$ was made the root subscriber and all the contacts of $S_2$ (not shown in figure) were added. Using this logic, we approximated a continuous data source for the purpose of this T-Check investigation. All of this logic was implemented in the OGSATCheckDriver subcomponent of the Java application.
- If the contact information of a particular subscriber is already stored, it should not be stored again. For example, in Figure 11 subscriber $S_8$ is a contact for both $S_4$ as well as $S_7$. The contact information of $S_8$ will already have been stored when the program is processing all the contacts of $S_4$. Therefore, it should be excluded and not stored again when the program is processing the contacts of $S_7$. We achieve this by storing a list of contacts whose information has already been visited and stored before.
- The Contact Graph Builder application should have the capability to resume from a particular node in the graph. For example, if we stop the application when it is processing the contacts of $S_4$, the program should start processing the contacts of $S_4$ when it resumes. This capability is achieved by keeping track of the current root subscriber.

One problem faced during this integration was the encoding format of the data obtained from the Web site. This data contained special characters that the OGSA-DAI service was unable to handle and store in any of the underlying databases. It was necessary to convert the raw data into UTF-8 encoding format before sending it to the OGSA-DAI data service for storage.

## 4.11 CREATING THE DATA CONSUMER AND THE DASHBOARD

We created a browser-based client for accessing all the information that was stored in the databases. This information was also accessed using OGSA-DAI data services, as shown in Figure 9. The browser-based client was an HTML-based Graphical User Interface (GUI) (see Figure 12) produced using a Java servlet deployed on a JBoss application server [JBossAS 07]. This servlet invokes the ConfigurableDataServiceProxy using a call-return connector. ConfigurableDataServiceProxy has references to all the available OGSA-DAI services. The dashboard user interface provides the following functionality:

- lists all the available data services along with their status (active or inactive)
- allows the user to activate/deactivate a service[13]
- provides the actual data from the all the active databases
- reflects changes to the actual data as new records are added to the databases

[13] The OGSA-DAI service corresponding to the “sacred” database cannot be inactivated.

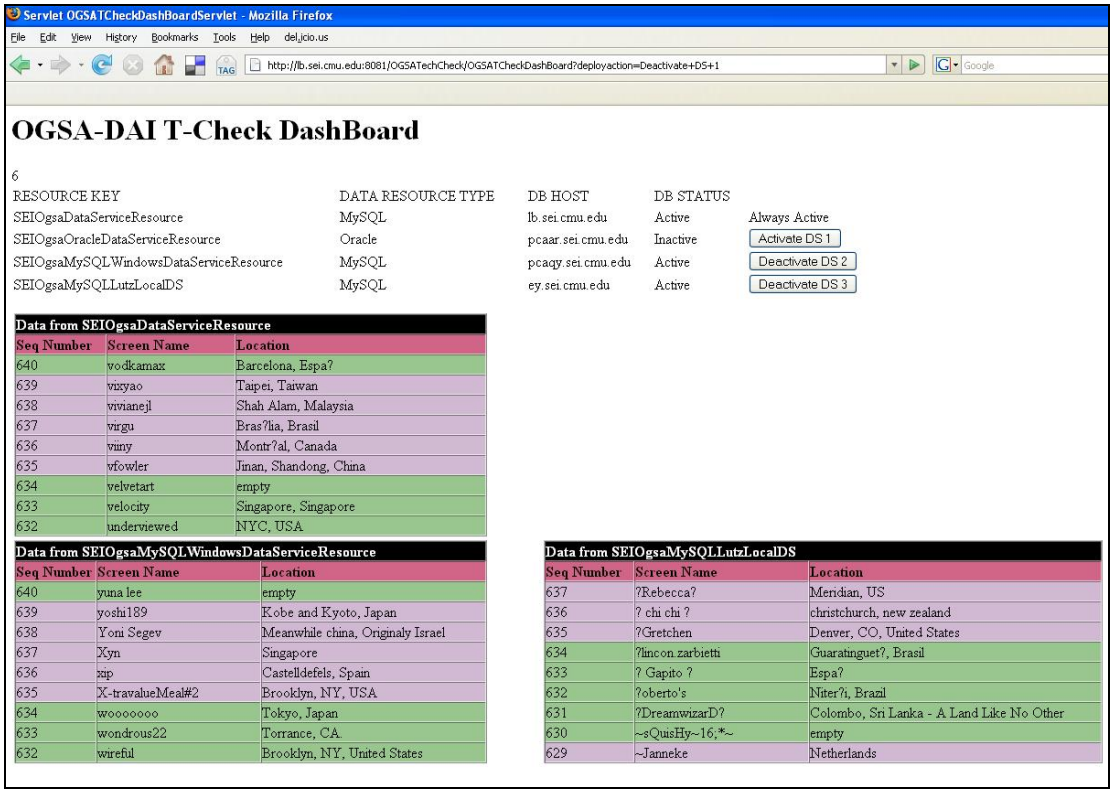


*Figure 12: Screen Capture of OGSA T-Check Dashboard GUI*

# 5 Evaluation and Experiences with OGSA

In this section, we present the results of evaluating the solution against the criteria.

## 5.1 RESULTS OF HYPOTHESIS 1

Hypothesis 1: **OGSA provides data storage and retrieval where the specific implementation or location of the data sources is transparent for raw data consumers and data processors.**

This first hypothesis was sustained because

- Data storage services can be used without the service consumer knowing the actual location of the database. The service consumer only needs to know the unique data resource identifier and the address of the Grid service container where the OGSA-DAI services are deployed. Using the unique identifier, the service consumer obtains a handle to the actual data service from the GTK4 container using the generic service fetcher class[14] provided with the toolkit. In this case, we had prior knowledge of the resource identifiers for each service. In a real scenario, these unique service identifiers can be obtained by querying a discovery service or registry.
- The data services deployed are identical from a service-consumer point of view, even if their underlying implementations are different. In this case, different database products were used as data stores and exposed as OGSA-DAI services. The use of different database products was not visible to the service consumer, and they were unaware that the data was fragmented and stored in different data stores. As explained in Section 4.6, four different data services were deployed, and data was stored in a round-robin fashion using the available services. In our case, the choice of which data service to use was based on a simple criterion because the services were used in a round-robin fashion. In a real-life scenario, service selection could be more complex and based on different criteria.

Although it might seem obvious, it is important to note that in our T-Check investigation, data could be stored in any of the databases because all they shared a schema that was created at design time. This circumstance may not hold true in all cases, especially in a dynamic service-oriented environment where database schemas are created at runtime. However, in a dynamic environment it is possible to create the database schema on the fly before storing the actual data into the data store. Another important factor to consider is the data format. The service and the underlying data store should be able to support the appropriate data-encoding format to avoid data corruption.

[14] A generic service fetcher class creates proxies for managing communications with data services depending upon the OGSA-DAI distribution used to deploy the service. This information is deduced by accessing namespaces within the service's WSDL description, which is accessed via its URL.

The following is a more detailed description of some of the additional findings that are relevant to this discussion.

### 5.1.1 OGSA-DAI is Slower Than Other Similar Technologies for Data Access and Retrieval

As explained in Section 4.4, OGSA-DAI is document-oriented, meaning that any communication between layers in the architecture is done via documents in XML format. Using a document-oriented architecture provides platform and language independence and provides a single, data-store-independent interface to a data service. It is understandable that this flexibility has costs and tradeoffs associated with it, and we were interested in objectively evaluating one of these trade-offs. We chose response time of the service as our evaluation criteria because it is a quality attribute applicable to any service. In this case, we decided to evaluate the amount of time the service takes to perform an *insert* database operation.

Having previously used other data access and retrieval technologies, we were curious to compare the response time of OGSA-DAI with them. Given the resource and time constraints, it was not possible to evaluate all the technologies available. We decided to compare OGSA-DAI services with two other standard data access mechanisms, namely JDBC and Enterprise Java Beans (EJB) [JDBC 07, EJB 07]. The following are our reasons for choosing these two technologies:

- Both JDBC and EJB are commonly used mechanisms for performing database interaction.
- The implementation of the OGSA-DAI framework is done in Java. Hence, we felt comparing OGSA-DAI with two other Java-based technologies was logical.
- Most of our system was implemented using Java-based technologies; hence, it was easier to compare it with other Java-based technologies. Moreover, we had an existing setup of the JBoss application server that could be used to deploy EJBs.

Our evaluation was based on empirical data collected by profiling the system at runtime. The profiling was done using timing logs that were recorded in a file. Figure 13 shows the three setups that were used. The same data and underlying database tables were used in each setup. For each case, data for two important timings was gathered: (1) the time taken by the Web service to fetch raw data from the online Web site and (2) the time taken to perform a single database operation. The database operation performed in each case was to insert new records containing the contact information into the database. The following are some of the steps that we took to keep the data consistent across the three setups:

- The same database instance running on the same physical machine was used. The database tables and indices were also reinitialized to create the same base configuration for each test.
- The tests were executed at almost the same time of the day. The first two tests (OGSA-DAI and JDBC) were conducted on the same day and the final test (EJB) was conducted on a different day but around the same time of the day.

- The tests were run at a time when the least internal network activity was expected[15] to minimize the effect of varying network traffic.
- As the highlighted rectangles in Figure 13 show, the only change for each setup was the data access mechanism.
- Data was fetched for the same root user of the external Web site in all the three cases, ensuring that we were dealing with similar, if not identical, datasets in all three cases.[16]

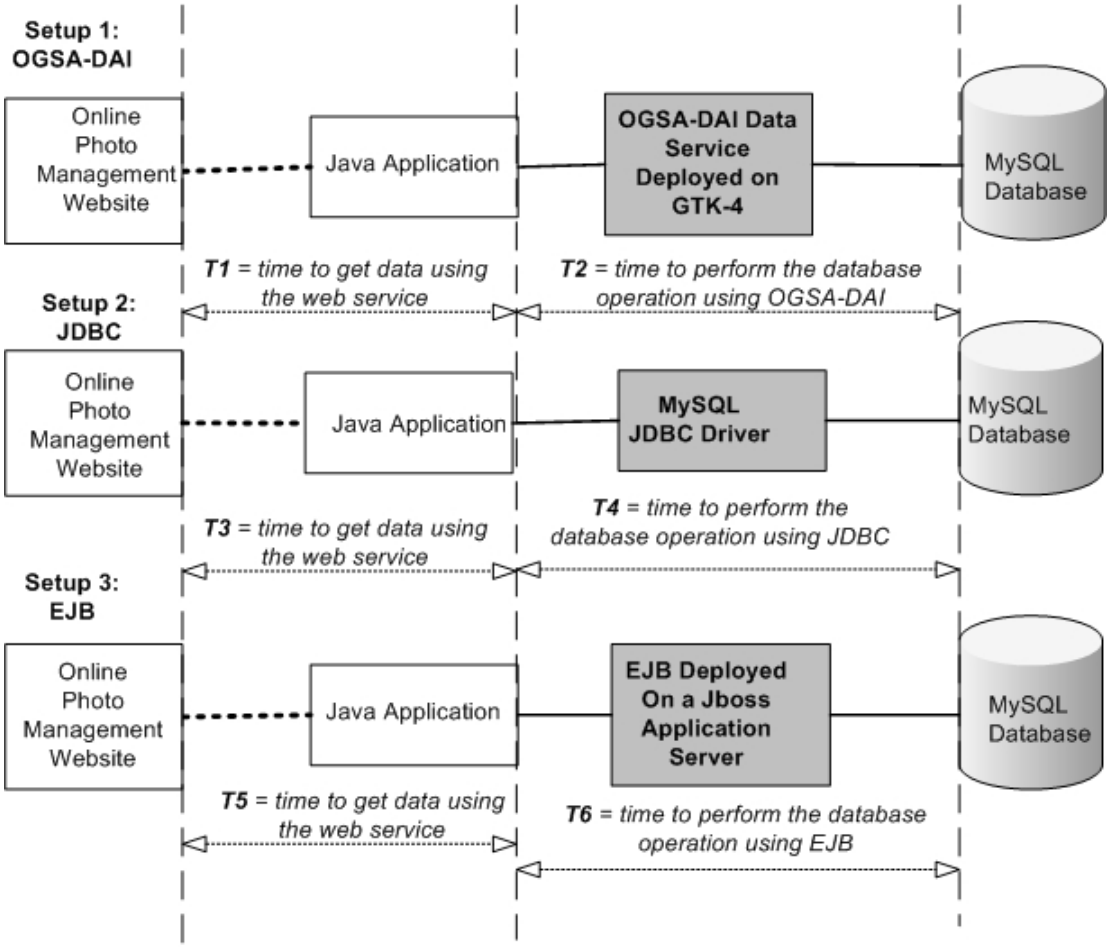


*Figure 13: Experiment Setups for Comparison*

[15] All the tests were run during evenings on weekends when the intranet network usage is the lowest. However, this does not guarantee the lowest amount of traffic inside the intranet.

[16] It is possible that there could be slight changes in the number of contacts for one or more users.

Table 7 shows the summary and comparison of the three tests that were conducted. The results shown in this table are consistent with our hypothesis that OGSA-DAI services are slower than their contemporary counterparts, JDBC and EJB, are.

*Table 7: Timing Data for Web Service and Three Different Data Access Mechanisms*

| | OGSA-DAI | JDBC | EJB |
|---|---|---|---|
| Total number of records fetched from the data service | 7128 | 7128 | 7128 |
| Average time taken to obtain data from the web service | 272 ms | 291 ms | 475 ms |
| Average time to store data into the database | 88 ms | 0.93 ms | 4 ms |

There could be several reasons for this behavior. As already explained, we suspect the primary reason for this is the document-oriented architecture of OGSA-DAI. A document-oriented architecture requires more steps, such as parsing documents. Both JDBC and EJB communicate using lower-level mechanisms; hence, they are faster. In addition, OGSA-DAI is a relatively new technology. JDBC and EJB are stable and mature technologies that have been widely adopted for distributed computing for several years. Both of those technologies have been optimized for best possible performance; OGSA-DAI has yet to see such a change.

In spite of having lower performance, OGSA-DAI services have advantages, especially for applications where flexibility is a bigger concern than performance. As we have seen, OGSA-DAI provides flexibility at the data access layer. OGSA-DAI services can be useful for service consumers looking more for loose coupling at the data access layer than strong performance.

### 5.1.2 Predicting and Controlling the Quality of Service when Using External Services May Not Always Be Possible

Another important understanding about service and Grid systems is supported by all the execution time tests we conducted. We found that the time to fetch the data from the external Web services varies dramatically. Again, this is not a new or unexpected observation, but it is an important one. Variable latency and sporadic failures are not exceptions in a large distributed environment; they are the rule. Some delay at the service provider's end, network latency, or a combination of these factors can cause these failures. Although it is important to understand all of the reasons behind these deviations, we will concentrate on another aspect of this problem—the lack of control over external services.

One fundamental consequence of Grid systems and service-oriented systems is lack of centralized control, as can be seen in the virtual organizations explained in Section 2.1. A lack of centralized control means that the service consumer has to trust the service provider and hope to get the best possible service. However, some elements are outside the control of the service provider, such as network latencies. Service level agreements (SLAs) are a common mechanism for implementing this mutual agreement between service provider and service consumer. Other techniques, such as data caching and proper service interface granularity, can be used to reduce the network traffic. Again, achieving control is difficult when the external services and network are not controlled by the service provider or the service consumers, which is generally the case for large, distributed systems.

### 5.1.3 Authentication Mechanisms for Web Services are Still Not Standardized

We had to understand and implement an authentication process before we could use the Web services APIs provided by Flickr. As mentioned before, Flickr allows any developer to use its Web services to develop custom, noncommercial applications. Before using these Web services, the developer must subscribe to Flickr.[17] Because Flickr uses Yahoo's authentication mechanism, any existing Yahoo subscriber can authenticate with Flickr using their Yahoo user ID and password. A developer who does not have a Yahoo account will have to sign up for one to obtain a user ID[18] and password. This Yahoo user ID and password can be used to authenticate a user into Flickr. Then the developer has to register an application with Flickr.

The Flickr authentication mechanism for third-party applications varies with the type of client application—desktop-based, web-based, or mobile application. The application type needs to be specified at registration time. In our case, we followed the authentication mechanism for desktop-based application. After a successful registration of an application, Flickr provides a unique API key and a shared secret. These interactions are shown in Figure 14. Both user and application registrations are only performed once.

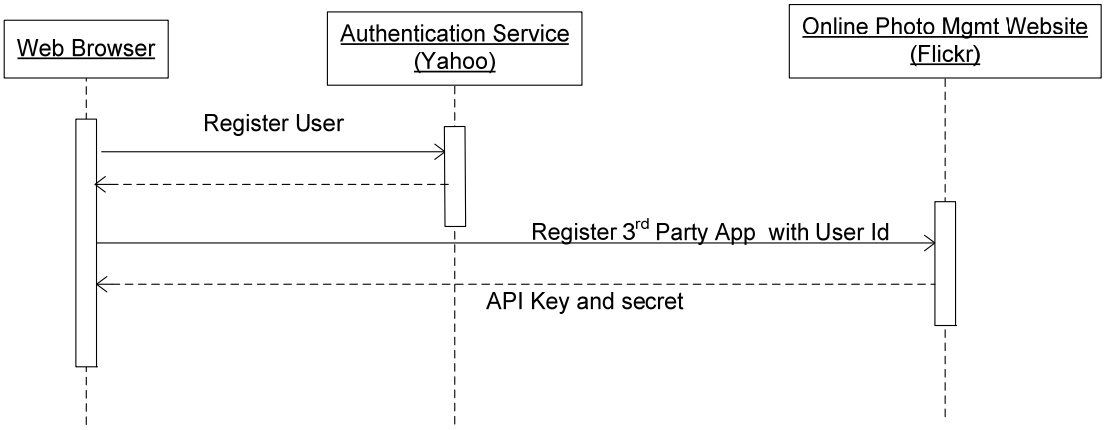


*Figure 14: Sequence of User and Application Registrations with Yahoo and Flickr*

Once both registration processes are completed, the following steps are necessary to authenticate and obtain data (see Figure 15):

1. The API key and shared secret combination are used to register a new session with Flickr. Upon valid session registration with Flickr, the Java application obtains a unique session identifier or token called a *frob*. This token is temporary and expires after a limited amount of time, after which a new token must be obtained for a new session.
2. The next step is to create an authentication URL, which is the login link. The login URL requires the API key, frob, permissions (read/write/delete), and API signature. The Java toolkit we used had already implemented the logic of creating the authentication URL. We just had to provide the required input.

[17] The basic version of subscription to Flickr is free.

[18] The Yahoo user ID is different from the Flickr user ID. The Yahoo ID is used for authentication purposes only.

3. The authentication URL is manually copied and pasted into a new Web browser window.[19] On receiving the request, the authentication server (in this case Yahoo) challenges back by prompting the user for the Yahoo user ID and password.
4. Once the user provides the correct user ID and password, the server prompts for a user consent page. The consent page describes the terms and conditions.
5. The user grants consent. We assume that the Yahoo server shares this session information with the Flickr server in some way that is internal and implementation-specific.
6. Finally, the Java application can invoke Web services to get data from the Flickr servers.

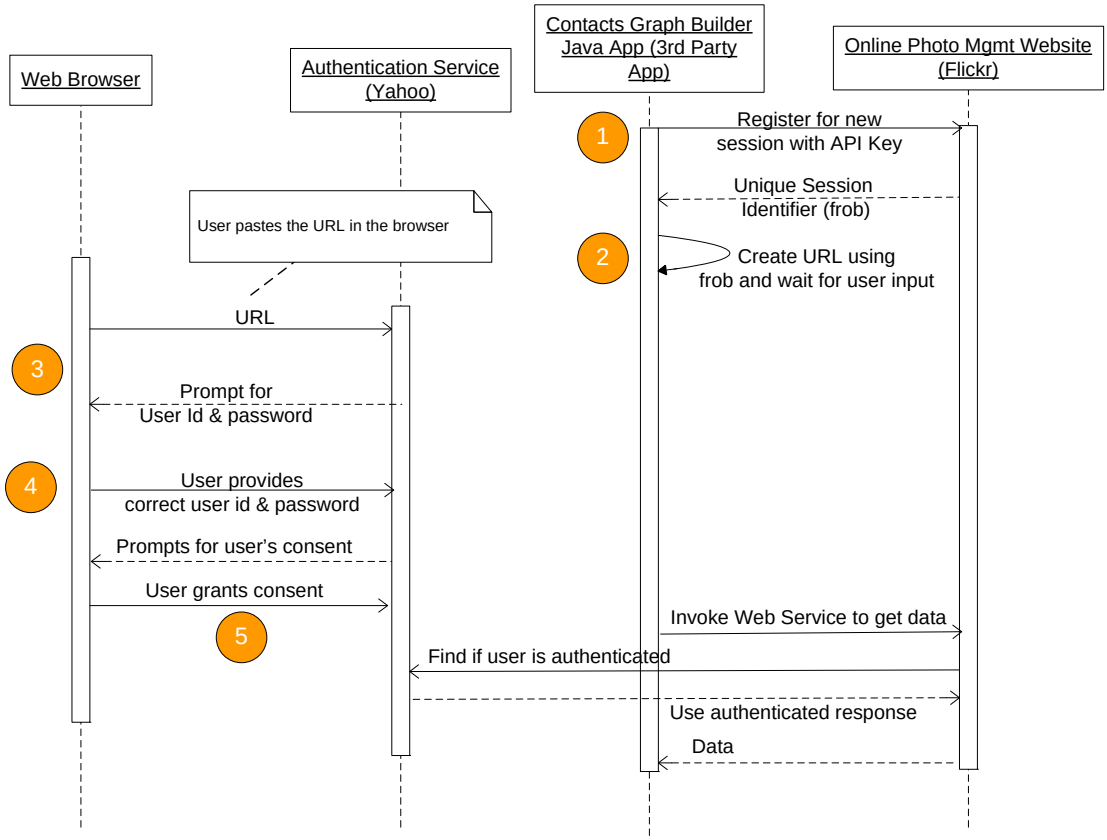


*Figure 15: Sequence of Activities for Authentication*

Even though Flickr provides standard Web services interfaces, we spent a considerable amount of time to understand its authentication mechanism. At the time of selecting the Web services, we did not consider this. We assume that the authentication mechanism that we have described is based on concepts similar to the emerging Security Assertion Markup Language (SAML) standard [SAMLTechOverview 07]. However, we did not find any references that support this assumption. This area of active research proved to be a problem in our small investigation.

[19] In a production scale application, this process should always be automated. We did not automate this process due to a lack of time.

Any consumer of external Web services should evaluate security requirements and constraints. In our case, had we implemented the Contact Graph Builder as a Web-based application instead of a Java application, the manual cut-and-paste (Step 3 on page 33) would not have been necessary.

## 5.2 RESULTS OF HYPOTHESIS 2

Hypothesis 2: **OGSA allows developers to add new data stores (databases) and remove existing data stores at runtime without affecting the overall operation of the system**.

The second hypothesis was also sustained because

- New data services were easily deployed and exposed without affecting the existing data services and the overall operation of the system. OGSA-DAI provides configurable data services that can be deployed at runtime. The only component that was aware of new service deployments was the DataServiceScheduler. The distribution of data was automatically handled by this component. In a more realistic architecture, this responsibility can be assigned to a similar broker component that takes care of managing data services. This broker component can be implemented by either the service provider or the service consumer. In this T-Check investigation, it was implemented as a component on the service consumer end. In our case, it did not make a big difference from an implementation point of view because we played both service provider and service consumer roles. In the real world, service provider and consumers are often different entities.
- We assume a graceful degradation scenario when existing services are removed. When an existing OGSA-DAI data service is removed, in other words, all the data in the underlying data store is no longer available to the service consumer. However, this data is not lost as long as it is not actually deleted from the underlying database.

### 5.2.1 It is No Surprise that Testing Distributed Applications is Complicated

An issue that required considerable debugging and research was related to the invocation of OGSA-DAI services from a remote machine. At the time of installation, we had tested invoking OGSA-DAI services only from the same machine on which the Globus container was running. We found that invoking the services from a remote machine did not work. OGSA-DAI provides a client toolkit that includes the necessary libraries required at compile time and runtime for any application that uses OGSA-DAI data services. We were able to compile the application successfully, but we received an End Point Reference (EPR) exception at runtime when we tried to invoke an OGSA-DAI data service [WS-Addressing 04]. The specific exception message on the server side was “The WS-Addressing To request header is missing exception.” The remote client failed to read the correct `client-config.wsdd` file containing the necessary handlers that automatically put the WS-Addressing headers into any message sent from the client to the OGSA container. This problem was resolved by adding the `client-config.wsdd` to the runtime classpath of the client application.

Although the solution was straightforward, we spent considerable effort to find the cause and devise the solution because the exception stack trace at the client and server sides did not provide enough clues. Because we were acting as service consumer and service provider, we had access to both exception stack traces. This circumstance might not be the case in large-scale Grid systems.

### 5.2.2 OGSA-DAI Data Services are Good Candidates for Infrastructure Services

Infrastructure services are lower level, generic services that are not business specific, such as data storage, data translation, and security. Based on our experience with service-oriented architecture (SOA), we feel that OGSA-DAI data services are more suitable as infrastructure services because they are not business specific. For example, a properly designed OGSA-DAI data service can be used by a command and control (C2) system or the customer relationship management (CRM) system of a large enterprise. An account creation service is business specific because it is designed to be used only in the context of enterprise systems; as a result, it may not be useful for a C2 system.

Ideally, data service consumers should be abstracted as much as possible from the implementation details of the data service. An abstraction layer on top of OGSA-DAI data services would be responsible for managing and coordinating various OGSA-DAI data services and performing other nonspecific business activities. Ideally, this layer should be the only interface visible to consumers of the data services.

In our T-Check implementation, the components DataServiceScheduler and ConfigurableDataServiceManager provide similar infrastructural capabilities. In a production environment, they can be completely decoupled from the actual business logic. In our implementation, we used four different data services; one of the services was considered “always alive.” In a real-life system, there should be multiple redundant “always alive” services to increase dependability. We decided to implement a simple round-robin scheduling policy. A more complex system could implement a policy that supports finding and replacing existing services dynamically. For example, if one of the underlying databases failed or ran out of space, the infrastructure layer could locate another functional OGSA-DAI data service and use it. These infrastructural responsibilities, such as switching services and adding new services, can be easily allocated to another layer. This arrangement not only reduces the burden on service consumers but also ensures that infrastructure activities are standardized and localized in one layer. Consequently, many different service consumers can reuse the functionality in a standard way. Ideally, this layer should also handle the dynamic allocation and de-allocation of various data resources, making the dynamism transparent to the end user of the data service.

# 6 Conclusions and Request for Feedback

Our overall experience using the T-Check approach for the evaluation of OGSA for data management was positive. We feel that Grid technologies have great potential, especially for organizations that are moving towards a service-oriented environment. However, given the vastness of the Globus Toolkit, it is not easy to evaluate the full technology objectively and in detail. This T-Check investigation promotes understanding of the fundamental concepts of Grid computing and OGSA and demonstrates how OGSA-DAI can be used to deal with heterogeneous data repositories. OGSA-DAI provides an implementation approach for creating data specific infrastructure services in a service-oriented environment.

The ISIS team that is investigating OGSA and other technologies using the T-Check approach is interested in feedback from and collaboration with the communities that are considering technologies for service-oriented environments. If you want to provide feedback or discuss collaboration, contact isis-sei@sei.cmu.edu.

# Appendix A Data Service Resource Properties File

As explained in Table 4 on page 17, a data service resource associates a physical data store with the OGSA-DAI data service resource that can then be associated with an OGSA-DAI data service. The characteristics of the actual data service resource are required to deploy a new data service resource. OGSA-DAI requires these characteristics to be specified in a properties file used during the deployment of a data service resource.

What follows is one of the four properties files that was used in the T-Check examination. The entries shown in boldface were modified. Similar files were used for the other three services.

```
##
## Data service resource configuration.
##

## 1-Select the name of the data service resource.
##   The default is "DataServiceResource".
##   You can change this if you want.

dai.resource.id=SEIOgsaDataServiceResource

## 2-Select the type of the data resource that forms the data
service
##   resource.
##   Remove the hash (#) from the front of the desired type.
##   Only remove one hash!

dai.data.resource.type=Relational
# dai.data.resource.type=XML
# dai.data.resource.type=Files
# dai.data.resource.type=MultiResource

## 3-Provide information about the data resource. This includes:
##     Product name, vendor and version (optional)
##     Database URI (required).
##     Database driver class name (required).
##
##   Remove the hash (#) from the front of the values
corresponding
##    to your data resource...
##
```

```
##   3A-Remove the hash (#) from these values if you are
deploying a
##      MySQL data resource
##      Then provide the URI and modify the other values if
required.

dai.product.name=MySQL
dai.product.vendor=MySQL
dai.product.version=4.0.2
#dai.data.resource.uri=jdbc:mysql://HOST:PORT/PATH/TO/DATABASE
dai.data.resource.uri=jdbc:mysql://lb.sei.cmu.edu:3306/ogsa
dai.driver.class=org.gjt.mm.mysql.Driver

##   3K-Remove the hash (#) from these values if you are
deploying a
##      another type of data resource
##      Then provide the URI and modify the other values if
required.

# dai.product.name=???
# dai.product.vendor=???
# dai.product.version=???
# dai.data.resource.uri=???
# dai.driver.class=???

## 4-Enter the initial credential that users need to provide to
##   access the data resource.
##   If no credentials need to be provided then leave blank.

dai.credential=

## 5-Enter the database username and password that will be used
to log
##   into the database.
##   If there is no username or password required or these can be
null
##   then leave empty.

dai.user.name=****
dai.password=****
```

# Appendix B Acronyms and Initialisms

| Acronym | Description |
|---|---|
| API | Application Programming Interface |
| C2 | Command and Control |
| CPU | Central Processing Unit |
| CRM | Customer Relationship Management |
| DNS | Domain Name System |
| DoD | Department of Defense |
| EJB | Enterprise Java Beans |
| EPR | End Point Reference |
| HTML | Hypertext Markup Language |
| HTTP | Hypertext Transfer Protocol |
| HTTPS | Hypertext Transfer Protocol Secure |
| GTK4 | Globus Toolkit 4 |
| GUI | Graphical User Interface |
| J2EE | Java Enterprise Edition |
| JDBC | Java Database Connectivity |
| OGSA | Open Grid Services Architecture |
| OGSA-DAI | Open Grid Services Architecture – Data Access and Integration |
| PHP | Hypertext Pre-Processor |
| QoS | Quality of Service |
| REST | Representational State Transfer |
| SAML | Security Assertion Markup Language |
| SDK | Software Development Kit |
| SLA | Service Level Agreement |
| SOA | Service-Oriented Architecture |
| TCP/IP | Transmission Control Protocol/Internet Protocol |
| URL | Universal Resource Locator |
| VO | Virtual Organization |
| WSDL | Web Services Interface Definition Language |
| WS-I | Web Services Interoperability |
| WSRF | Web Service Resource Framework |

# References

*URLs are valid as of the publication date of this document.*


**[Apache 07]**
The Apache Ant Project. *Welcome*. http://ant.apache.org/ (2007).

**[Childers 06]**
Childers, Lisa & Sotomayor, Borja. *Globus Toolkit 4 Programming Java Services.* San Francisco, CA: Morgan Kaufmann, 2006.

**[EJB 07]**
Sun Microsystems, Inc. *Java Platform, Enterprise Edition (Java EE), Enterprise JavaBeans Technology*. http://java.sun.com/products/ejb/ (1994-2007).

**[Fielding 00]**
Fielding, R.T. "Architectural Styles and the Design of Network-Based Software Architectures." PhD diss., University of California, Irvine, 2000. http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

**[Flickr 07]**
Yahoo, Inc. *About Flickr*. http://www.flickr.com/about (2007).

**[FlickrAPI 07]**
Yahoo, Inc. *Flickr Services*. http://www.flickr.com/services/api/ (2007).

**[Foster 01]**
Foster, Ian, et al. *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*. http://www.globus.org/alliance/publications/papers/anatomy.pdf (2001).

**[Foster 02a]**
Foster, Ian. *What is the Grid? A Three Point Checklist*. http://www-fp.mcs.anl.gov/~foster/Articles/WhatIsTheGrid.pdf (2002).

**[Foster 02b]**
Foster, Ian, et. al. "Grid Services for Distributed System Integration." *IEEE Computer 35*, 6 (June 2002): 37–46.

**[Foster 04]**
Foster, Ian & Kesselman, Carl. *The Grid: Blueprint for a New Computing Infrastructure.* San Francisco, CA: Morgan Kaufmann, 2004.

**[Foster 06]**
Foster, I. "Globus Toolkit Version 4: Software for Service-Oriented Systems." 2–13. *Proceedings of the IFIP International Conference on Network and Parallel Computing (NPC 2005) (Lecture Notes in Computer Science, 3779).* Beijing, China, November 30-December 3, 2005. Berlin, Germany: Springer-Verlag, 2006.

**[GlobusAlliance 06]**
The Globus Alliance. *Welcome to Globus*. http://www.globus.org/ (2006).

**[HP 05]**
Hewlett-Packard Inc. *British Columbia Cancer Agency Improves Patient Care with Distributed Storage Grid*. http://h71028.www7.hp.com/ERC/downloads/4AA0-2779ENA.pdf (2005).

**[JBossAS 07]**
JBoss.org. *JBoss Application Server*. http://labs.jboss.com/portal/jbossas (2007).

**[JDBC 07]**
Sun Developer Network. *Java SE – Java Database Connectivity (JDBC)*. http://java.sun.com/javase/technologies/database/ (1994-2007).

**[Lewis 04]**
Lewis, Grace A. & Wrage, Lutz. *Approaches to Constructive Interoperability* (CMU/SEI-2004-TR-020, ADA431067). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2004. http://www.sei.cmu.edu/publications/documents /04.reports/04tr020.html

**[Lewis 05a]**
Lewis, Grace A. & Wrage, Lutz. *A Process for Context-Based Technology Evaluation* (CMU/SEI-2005-TN-025, ADA441251). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2005.
http://www.sei.cmu.edu/publications/documents/05.reports/05tn025.html

**[Lewis 05b]**
Lewis, Grace A; Morris, Ed; O'Brien, Liam; Smith, Dennis; & Wrage, Lutz. *SMART: The Service-Oriented Migration and Reuse Technique* (CMU/SEI-2005-TN-029, ADA441900). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2005.
http://www.sei.cmu.edu/publications/documents/05.reports/05tn029.html

**[Lewis 06]**
Lewis, Grace A. & Wrage, Lutz. *Model Problems in Technologies for Interoperability: Web Services* (CMU/SEI-2006-TN-021, ADA454363). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2006.
http://www.sei.cmu.edu/publications/documents/06.reports/06tn021.html

**[Myer 04]**
Myer, Thomas. *Grid Watch: The Importance of Being Stateful*. http://www-128.ibm.com/developerworks/grid/library/gr-wsrf.html (2004).

**[NWS 07]**
National Weather Service. *National Digital Forecast Database XML Web Service*. http://www.weather.gov/xml/ (2007).

**[OGF 07]**
Open Grid Forum. *Success Stories & Case Studies*. http://www.ogf.org/UnderstandingGrids/ggf_resource_caseStudies.php (2007).

**[OGSA-DAI 07a]**
OGSA-DAI.org. *The OGSA-DAI Project*. http://www.ogsadai.org.uk/ (2007).

**[OGSA-DAI 07b]**
OGSA-DAI.org. *OGSA-DAI Architecture*.
http://www.ogsadai.org/documentation/ogsadai-wsrf-2.2/doc/background/architecture.html
(2007).

**[OGSA-DAI 07c]**
OGSA-DAI.org. *Interacting with Data Service Resources*.
http://www.ogsadai.org/documentation/ogsadai-wsrf-2.2/doc/interaction/index.html (2007).

**[SAMLTechOverview 07]**
Organization for the Advancement of Structured Information Standards (OASIS). *Security Assertion Markup Language, (SAML) V2.0*.
http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security#samlv20 (1993-2007)

**[Wallnau 01]**
Wallnau, Kurt; Hissam, Scott; & Seacord, Robert. *Building Systems from Commercial Components*. New York, NY: Addison-Wesley, 2001.

**[WS-Addressing 04]**
Worldwide Web Consortium. *Web Services Addressing*.
http://www.w3.org/Submission/ws-addressing/ (2004).

**[WS-I 06]**
Web Services Interoperability Organization. *About WS-I*. http://www.ws-i.org/ (2006).

**[WSRF 06]**
OASIS. *Web Services Resource Framework*.
http://docs.oasis-open.org/wsrf/wsrf-primer-1.2-primer-cd-02.pdf (2006).

**[Zhang 05]**
Zhang, Liang-Jie; Chung, Jen-Yao; & Zhou, Qun. *Developing Grid Computing Applications, Part 1*. http://www-128.ibm.com/developerworks/library/gr-grid1/ (2005).

# REPORT DOCUMENTATION PAGE

*Form Approved*
*OMB No. 0704-0188*



| 1. AGENCY USE ONLY | 2. REPORT DATE | 3. REPORT TYPE AND DATES COVERED |
|---|---|---|
| (Leave Blank) | April 2007 | Final |

| 4. TITLE AND SUBTITLE | 5. FUNDING NUMBERS |
|---|---|
| T-Check℠ for Technologies for Interoperability: Open Grid Services Architecture (OGSA)—Part 1 | FA8721-05-C-0003 |

6. AUTHOR(S)

Soumya Simanta, Grace A. Lewis, Lutz Wrage

| 7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) | 8. PERFORMING ORGANIZATION REPORT NUMBER |
|---|---|
| Software Engineering Institute<br>Carnegie Mellon University<br>Pittsburgh, PA 15213 | CMU/SEI-2007-TN-016 |

| 9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) | 10. SPONSORING/MONITORING AGENCY REPORT NUMBER |
|---|---|
| HQ ESC/XPK<br>5 Eglin Street<br>Hanscom AFB, MA 01731-2116 | |

11. SUPPLEMENTARY NOTES

| 12A DISTRIBUTION/AVAILABILITY STATEMENT | 12B DISTRIBUTION CODE |
|---|---|
| Unclassified/Unlimited, DTIC, NTIS | |

13. ABSTRACT (MAXIMUM 200 WORDS)

Many current technology approaches exist for building systems that have interoperability requirements. This report investigates Open Grid Services Architecture (OGSA), one of the many technologies for accomplishing interoperability, using the T-Check technique. A T-Check is a simple and cost-efficient way to understand what a technology can and cannot do in a specific context. This report describes a T-Check exploration of the feasibility of using OGSA in the context of data management, finding that OGSA (a) provides data storage and retrieval where the specific implementation of the data store implementation is transparent and (b) allows addition or removal of data stores at runtime without affecting system operation. This report is part one of a two-part investigation; part two will look at OGSA in the context of load distribution.

| 14. SUBJECT TERMS | 15. NUMBER OF PAGES |
|---|---|
| T-Check, OGSA, Grid computing, interoperability | 52 |

16. PRICE CODE

| 17. SECURITY CLASSIFICATION OF REPORT | 18. SECURITY CLASSIFICATION OF THIS PAGE | 19. SECURITY CLASSIFICATION OF ABSTRACT | 20. LIMITATION OF ABSTRACT |
|---|---|---|---|
| Unclassified | Unclassified | Unclassified | UL |