

# Designing for Broadly Available Grid Data Access Services

Rob Baxter, Denise Ecklund, Aileen Fleming, Alan Gray,  
Brian Hills, Stephen Rutherford and Davy Virdee

edikt, National e-Science Centre, University of Edinburgh, Edinburgh UK

[www.edikt.org](http://www.edikt.org) [info@edikt.org](mailto:info@edikt.org)

## Abstract

The Data Access and Integration Working Group (DAIS-WG) within the Global Grid Forum (GGF) is writing the Grid Database Service Specification (GDSS). The GDSS defines a Grid Data Service (GDS) standard for accessing and integrating data stored in multiple types of data storage systems, such as relational databases, XML databases, and file systems. The edikt team at the National e-Science Centre is developing Eldas, a commercial quality implementation of core features defined by GDSS.

By product quality implementation we mean that Eldas is robust, well documented, easy to use and is based on a modular and extensible design. While a product quality implementation is a basic requirement for system adoption, it does not ensure broad use of the technology among the grid community. The extent of take-up is enhanced or dampened by factors such as platform independence, compatibility with different technologies, options for layered services and ease of deployment in a grid setting. In this paper, we describe the take-up issues and associated design decisions made by edikt to address them.

## 1 Issues to Broad Adoption

Data is a critical resource in any research endeavour. Legacy data, raw data and derived data are essential, and reliable tools to access and integrate such data are in high demand. Addressing these needs, the edikt team at the National e-Science Centre is developing Eldas, Figure 1, a commercial quality implementation of the GDSS. Traditionally the phrase “commercial quality software” means a software package is robust, well-documented, easy to use and designed to support evolution and the addition of new service features. When developing data access and integration software for the grid environment, these aspects of product quality (while necessary) were not sufficient to ensure broad adoption of the Eldas software. Working with application scientists in several disciplines, edikt identified four major issues that could constrain Eldas adoption and made design decisions to mitigate or remove these obstacles. We describe these issues and outline the edikt approach to resolving them.

### Issue 1 – Machine and DBMS independence:

Research scientists use different computer systems to perform research analysis. They also use different database management systems and

file systems to store their basic and derived data. If a GDS runs on only one operating system, its take-up is highly constrained. To cover multiple architectures, Eldas is implemented within J2EE Java environments which are machine independent. Tested environments include JBoss and Sun ONE AS. An Eldas design and implementation using Enterprise Java Beans is underway. Using J2EE as our development environment ensures broad availability for scientists using the grid. To support a broad range of database types, our design encapsulates database-specific code in a Data Access Component, leaving a reusable Eldas core that is common to the set of Java environments we have examined.

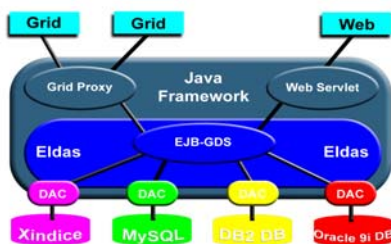


Figure 1: Eldas Conceptual Schematic

**Issue 2 – Dual compatibility:** Although web services differ from grid services in that they have neither state nor lifetime, the two are similar in many other respects. The separation of the Eldas business logic from the presentation layer allows service invocation using both grid and web services interfaces. This dual compatibility with both web services and grid services allows Eldas to serve a significantly larger user community.

**Issue 3 – Layered services:** The basic data access services provided by Eldas are GDSS-compliant. Several layered services are being designed and implemented, including support for annotating scientific data, detecting and resolving data value conflicts when integrating data from multiple databases and versioning and archiving scientific data. These services can be deployed and used with Eldas (or other GDSS-compliant implementations) in a mix and match fashion. Layered services increase the usefulness of the grid. This attracts more users, who bring requirements for new layered services, creating a synergistic feedback loop.

**Issue 4 – Ease of Use:** The GGF Open Grid Services Infrastructure (OGSI) specification defines properties and “interaction rules” that must be supported by all grid services. Without rules, grid services can not interact and the grid becomes useless. Developing services adhering to these requirements is a complex and error-prone task. To address this, edikt has created a fully automated installation and deployment process for Eldas. Our fully-documented, easy to install, deploy and use software will allow scientists to turn their applications in to widely available grid services.

Our Eldas design and implementation addresses these issues. In so doing, Eldas will make grid data services available to more scientists for less effort, making the grid a more effective environment for scientific research and collaboration.

### 1.1 Eldas

The Grid Database Service Specification (GDSS) [2][3], written by the DAIS working group of the GGF, defines a Grid Data Service (GDS) standard for accessing and integrating data stored in multiple types of data storage systems, such as relational databases, XML databases and file systems. The specification focuses on the functional requirements of data services for scientific applications [5]. It defines a data access and integration service “pipe” for

accessing external data management services. For example, GDSS supports passing SQL queries to relational database, queries expressed in XPath or XQuery to XML database systems, OQL queries to Object-oriented database systems and “read” and “write” operations to a file system.

The edikt team at the National e-Science Centre is developing Eldas – Enterprise Level Data Access Services – a commercial quality lightweight, robust and scalable implementation of core features of the GDSS, built around the Enterprise Java Beans (EJB) architecture [4]. For its initial implementations, edikt has chosen two free-to-use Enterprise Java application servers to maximise uptake of the software – JBoss [6] and Sun ONE Application Server [7].

Much of this paper presents our motivations in choosing the EJB architecture as an ideal platform in which to build grid data access services. We have been driven by the desire to make grid data services as simple to deploy and use as possible. The four key issues of platform independence, dual compatibility, layered services and usability form the cornerstones of the Eldas design.

### 1.2 Related work

Eldas is a robust implementation of the GDSS. Other reference implementations of the GDSS exist, such as the OGSA-DAI project [8]. At the time of writing we are unaware of other publicly announced implementations of the GDSS based on EJBs.

### 1.3 Structure of this paper

Section 2 presents an overview of Java Enterprise technologies and discusses their eminent suitability as a platform for developing grid data services. We cover the Enterprise Java Beans model (Section 2.2), discuss the use of EJBs for database access (Section 2.3) and review the concepts of web and grid services (Section 2.1). Section 3 presents our designs for Eldas and Section 4 offers our conclusions and pointers to further work.

## 2 J2EE – Platform of Choice for Grid Data Services?

This section provides a brief introduction to J2EE Java technologies and how they are used as web services and grid services. It also outlines a high-level architectural design view for implementing grid data services in an Enterprise Java Bean framework.

J2EE stands for the “Java 2 Platform, Enterprise Edition”, a technology designed to support applications that need to be scalable, available, reliable, secure, transactional and distributed. J2EE has become the *de facto* standard for developing multi-tier enterprise applications, by

having standardised, modular components and services. This allows many details of application behaviour to be created easily, without complex programming.

A typical, high level J2EE server-side architecture is shown in Figure 2:

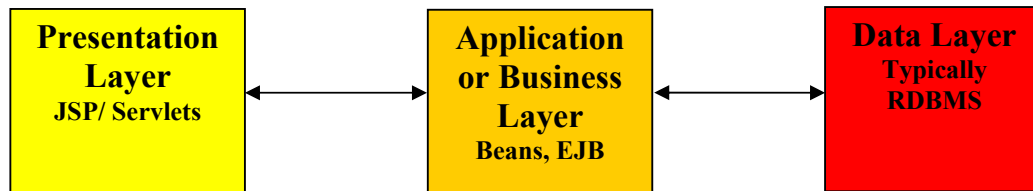


Figure 2: Layering in the J2EE Framework

The concept of layering is used to group together and separate components depending on their function:

- Presentation layer - contains web pages and user interfaces on the server;
- Application or business layer - all application code, EJBs;
- Data layer - the database.

A typical enterprise application will employ all of the above layers and components.

On this basis, edikt has developed Eldas as a set of services based on the GDSS which use the J2EE application model to access data resources. Enterprise Java Beans are used as server-side components in Eldas as their architecture enables the factory and service features of grid technology to be implemented easily. We discuss this more fully in Section 1.

### 2.1 Web and Grid services

Simply put, a web service is a way of exposing method(s) over the internet via interfaces which are both platform and language independent.

Typically, a client obtains information about an exposed method through a *Web Services Description Language* (WSDL) document. This then allows the client to communicate with the method via *Simple Object Access Protocol* (SOAP) and perform operation calls on the service using the Java API for XML based Remote Procedure Calls (JAX-RPC).

A web service is a *static* instance. That is, someone (a third-party) physically deploys the service on a server, and user/clients then decide if they wish to use this service. In contrast, a *grid service* is a service resource that is

deployed *dynamically*. That is, the service is not deployed *until* such time as the client requires it. The client chooses *which* service they require from a list available. It is also worth observing that a web service has the same lifetime as the container in which it is deployed – whereas a grid service can have a finite lifetime, distinct and (generally) shorter than that of the container. An additional distinction is that grid services are stateful, whereas web services are stateless operations. Grid services use WSDL and SOAP in the same way as web-services. Grid services currently use the Grid extensions to Web Service Description Language (G-WSDL), which is an extension of WSDL. The behaviours of grid services are defined in the OGISI Grid Service Specification [9][10].

### 2.2 Enterprise Java Beans – a scalable grid service architecture?

*Enterprise Java Beans* (EJBs) is a specification for Java built specifically for multi-tier web applications [4]. EJBs are an integral component of the J2EE platform, forming the basis of a distributed framework for J2EE. This distributed framework allows a client to make a call to the interface of a business object. The client communicates to the server object via a *stub* on the client which in-turn communicates this to a *tie* on the server object. A stub is also known as a *proxy* or a *surrogate*, while a tie is also known as a *skeleton* (confusingly a tie may also be referred to as a *proxy*).

The EJB framework is such that:

- EJB components are server side and written in Java, giving platform independence;
- EJB components contain *application code* (or *business logic*) only;

- the EJB container manages system level services such as security, transactions, life-cycle, threading and persistence of the components within the server;
- developers gain all the benefits of component based engineering.

EJBs come in a number of flavours, the most relevant of which is the *session bean*. Session beans are *transient* objects. A session bean created by a client usually exists only for the duration of a single client-server session. A session bean usually performs operations such as calculations or data transformations on behalf of the client. Session bean objects can be either *stateful* or *stateless*.

A *stateful* session bean maintains a *conversational state* across methods and transactions, i.e. *field values* of the bean need to be updated to maintain data consistency. Stateful beans know about their properties i.e. their *state*. Stateful beans are associated with a specific client. A *stateless* session bean represents work performed by a client. This work can be performed within a single method call or multiple method calls. It contains no information about its state. It can be thought of as an operator, only.

A second flavour of EJB is the *entity bean*. Entity beans are object representations of *persistent* data maintained in a permanent database. A primary key identifies each entity bean. Entity beans represent specific data or collections of data, such as a row in a database. An entity bean persists and survives as long as its data remains in the data base.

Both session and entity EJBs have similar behaviour when invoked. Both have an Object Factory, which instantiates a specific instance, or EJB Object, which in turn supplies a service to the user.

EJBs have to be deployed in a *container*, which supports the above services. Containers provide runtime support for J2EE applications. When a J2EE application is deployed, the deployment process installs each component in the appropriate container.

There are four containers in a typical application server hosting environment:

- EJB Container (server-side) manages the execution of all EJBs;
- Web Container (server-side) – manages execution of JSPs and Servlets;

- Application Client Container (client-side) – manages execution of all client components;
- Applet Container – Web Browser and Java plug-in together, or Web Browser and Java Runtime Environment (JRE).

### 2.3 Using EJBs for web and grid services

In Figure 3 we see how the web service API located in the presentation layer communicates with the business logic, which can take the form of beans and/or EJBs. Objects containing requests and results are exchanged between the client and server via SOAP.

In deploying grid services in the application server frameworks discussed above, one has the following model.

1. The client obtains a reference to a Grid Service Factory (GSF) from a Grid Service Registry (GSR).
2. To use a given service, the client connects to and communicates with the GSF via G-WSDL. This GSF is in fact a grid service itself since it uses G-WSDL extensions. It is not incorrect to see similarities between a GSF and web services, however, since a GSF will quite often be a *static* service deployed on the server.
3. The GSF now creates a Grid Service (GS) which allows the client to execute the methods that the GSF exposes.
4. The client now communicates with the GS using SOAP until the session is terminated. This grid service remains alive until the client has disconnected from the service or a previously agreed time-out elapses.

Here we have a clear correspondence between the EJB architecture and the behaviours of grid services and grid service factories. The EJB Object Factory behaves just as we expect a GSF to behave, creating instances of EJB Objects to perform defined tasks.

Within the application server framework, the fact that grid services are an extension of web services, and that there is a clear demarcation between presentation and business layers, means that exposing the application implemented in the EJBs can be done either way. Thus, data access services implemented this way can be accessed as either web services or grid services with no re-engineering of the application core required. This significantly

increases the potential user community for these services.

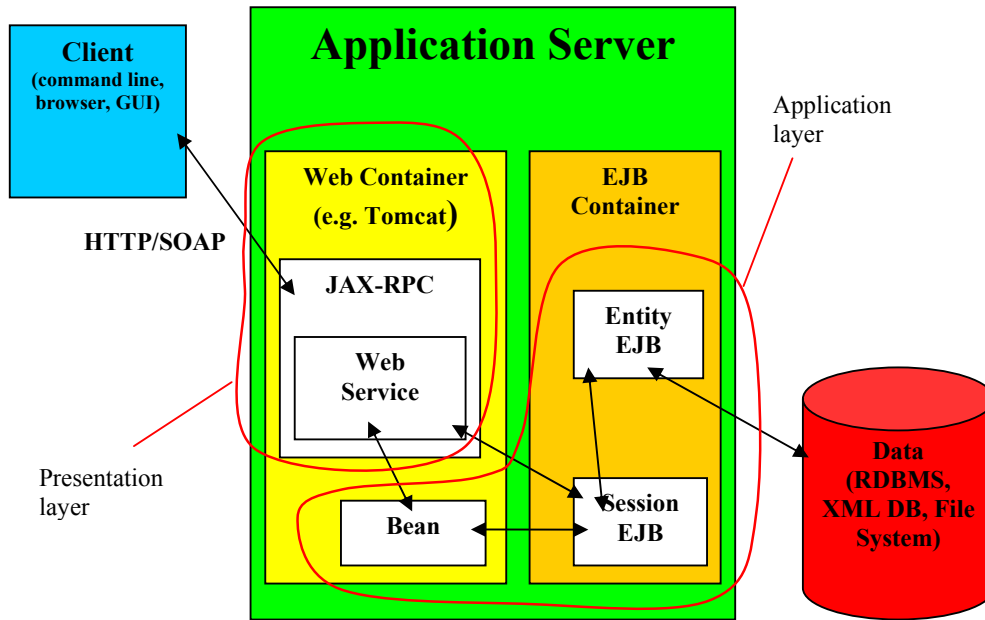


Figure 3: Web services and the J2EE framework.

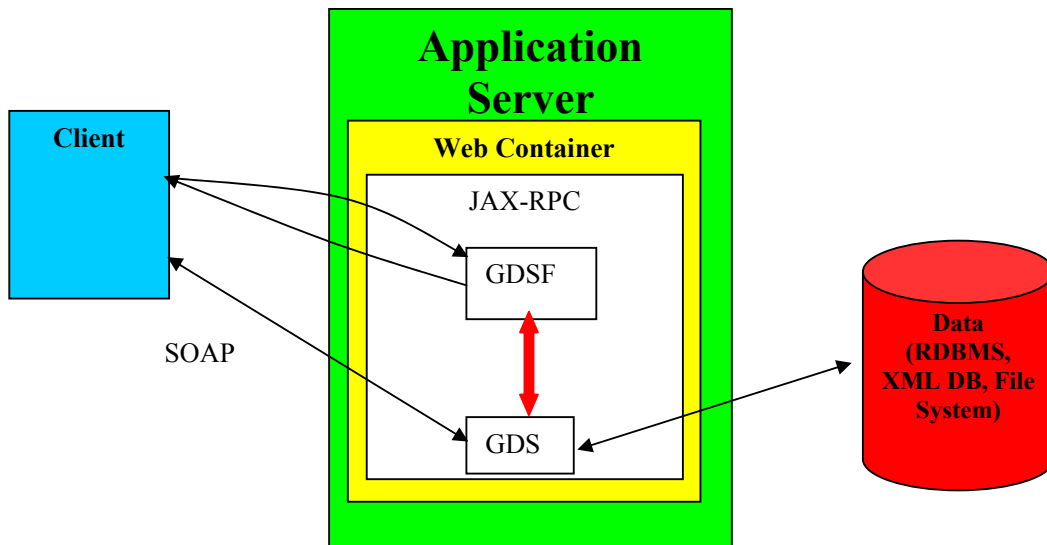


Figure 4: The OGSA-DAI Architecture.

### 3 The Eldas Design

The Grid Data Service Specification extends the concepts of the Grid Service Specification to allow database access, in much the same way as Enterprise applications described above. Figure 4 shows one way of implementing the grid data service architecture, using web containers and a

JAX-RPC application in which the grid services reside. This is the architecture used in the OGSA-DAI reference implementations of the GDSS [8].

As we have seen, grid services can be and are implemented using a web services framework, making use of all the features and functionality

of J2EE, i.e. web containers and application servers. There are many similarities between the way EJBs and grid services behave. An implementation of the GDSS interfaces in an EJB framework gives all the advantages of enterprise level applications to grid database access, namely:

- state management;
- data/object persistence;
- security;
- transactional scalability;
- tried and tested industry-strength robustness.

The high-level Eldas architecture is shown in Figure 5. We implemented the GDSS of G-WSDL port types as an EJB. A GDSF has the

same behaviour attributes as an EJB Object Factory and a GDS has the same behaviour attributes as an EJB Object. Additional interfaces needed to be added as a Proxy in the JAX-RPC layer to enable full grid service behaviour. This allows us to:

1. move all application code into the business layer, i.e. the EJB Container;
2. trim the functionality contained within the presentation layer down to an absolute minimum, i.e. a proxy within the JAX-RPC application.

The amount of functionality in the presentation layer is thus minimised through this use of EJBs. This architecture follows a *delegation pattern*.

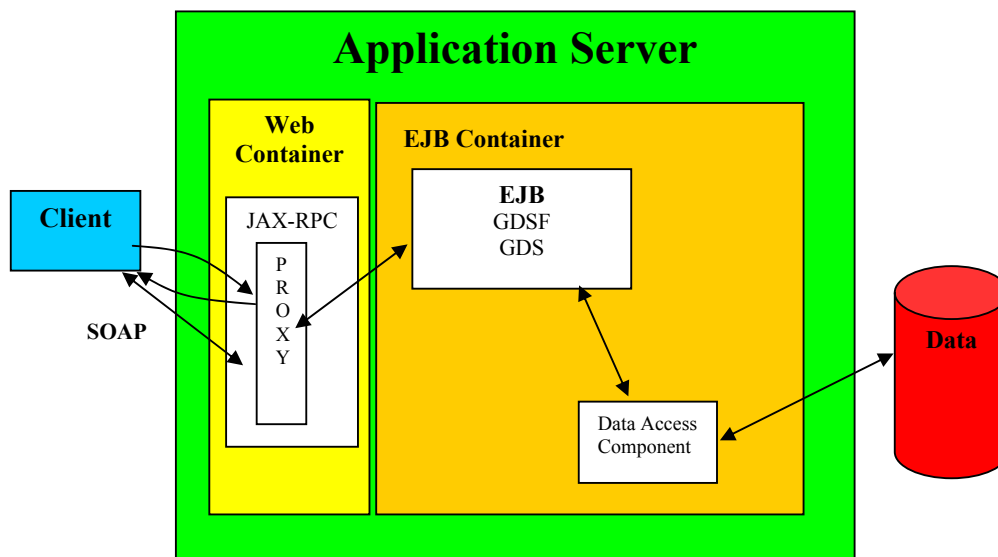


Figure 5: The Eldas architecture implemented in an EJB framework.

The primary reason for this is that the EJB framework has been designed as a scalable, high-performance container for “significant” application code, providing a more robust runtime environment for the execution of the grid database service code.

Much of the design of Eldas was based on well known patterns, most notably Business Delegate, Data Access Object and Singleton [11]. The concept of delegate classes is important for a number of reasons.

- They hide the implementation details of business service lookup/creation/access.
- They protect the business tier components from direct exposure to clients – the client doesn’t access the EJBs directly.
- They minimise coupling between presentation and business tiers.
- They reduce network traffic between client and business services.
- The business delegate handles exceptions from the business tier and generates application level exceptions

- these are easier for clients to handle. The delegate may also perform any retry or recovery operations necessary without exposing the client to the problem.
- They allow us to extend the FactoryServiceSkeleton and ServiceSekeleton classes of the OGSi framework. The EJBs cannot do this directly as they already extend other classes and multiple inheritance is neither desirable nor supported in Java.

In this way, the exposed interface of the EJB Object Factory, accessed via a delegated proxy, acts as the GDS Factory, creating an interface through which we can call the methods on the EJB Object. This remote object then acts as the actual Grid Data Service, accessed again via a delegated proxy in the presentation layer.

The Grid Data Service is implemented as a stateful session EJB. The EJB has been implemented this way so that only one bean is available for each client session, thus maintaining the conversational state with the client; in other words, they can have state or instance fields that can be initialised or changed by the client with each method invocation.

Unlike many EJB database applications, we do not use entity beans to access the database, since for each query produced by the client the overhead of invoking an entity bean each time would be very large on large datasets. Instead, we couple the GDS session bean to a data resource using the Data Access Component (DAC).

The DAC has been designed to allow access to various data resources depending on the driver used. The DAC has been implemented as a pluggable component, allowing changes to be made easily to the underlying implementation without affecting the development of the bean classes. Currently, Eldas supports access to several database management systems including:

- MySQL®<sup>1</sup>, an open source system from MySQL AB;
- DB2®<sup>2</sup>, from IBM Corporation;
- Oracle 9i™<sup>3</sup>, from Oracle Corporation;

---

<sup>1</sup> A registered trademark of MySQL AB.

<sup>2</sup> A registered trademark of IBM Corporation.

- Xindice, a freeware XML database system from the Apache Software Foundation, Inc.

### 3.1 Enhanced data services

The basic data access services provided by Eldas are GDSS-compliant. While these basic services are necessary for data sharing among collaborating scientists, basic services will not retain users over time. As scientists become acquainted with the grid, they will quickly envision new data services that will better support their work. To this end, the GDSS was defined with layered services in mind.

While designing Eldas, we worked with scientists in the areas of astronomy, particle physics and bioinformatics to better understand their long range goals in grid data services. Initially we assumed that different application areas would require different types of enhanced data services. Happily we were wrong. All of these areas (and we predict other areas such as earth sciences) have similar requirements for:

- annotating scientific data;
- detecting and resolving data value conflicts when integrating data from multiple data sources;
- maintaining integrity and consistency constraints over data stored in multiple data sources;
- transforming data representations, for all data types including binary data;
- managing versioned data;
- automatic archiving of raw and derived scientific data.

Users want to mix-and-match these services as needed, and we must support this requirement (even though such services are beyond the scope of the GDSS). Conceptually, enhanced data services are layered over Eldas. From an implementation perspective, it is advantageous to have factories that create tailored grid data services configured with exactly the set of required enhanced data services. Clearly such bespoke services will yield better performance. The current design includes complete Eldas functionality with each combination of enhanced services. It is possible that some enhanced data services use only a subset of the Eldas services. If so, a good componentised design will allow the unused basic services within Eldas to be removed from the tailored grid data service.

---

<sup>3</sup> A trademark of Oracle Corporation.

The ability to provide bespoke enhanced data services in a mix and match fashion increases the usefulness of the grid and will retain grid users as they become more sophisticated in their usage.

#### 4 Conclusions and further work

We believe that the architectural framework of the J2EE model provides an ideal platform upon which to build robust, scalable and easy-to-use grid data services.

In addition, our Eldas design and implementation addresses the four key issues blocking broad adoption of the Eldas technology. Hence, Eldas will make grid data services available to more scientists in a shorter time, making the grid a more effective environment for scientific research and collaboration.

Version 1.0 of Eldas is scheduled for completion in Autumn 2003. The Eldas software and full documentation will be made available to the UK e-Science community from the edikt project website, [www.edikt.org](http://www.edikt.org). It is our intention to track the evolving nature of the GDSS and ensure Eldas remains compliant with this specification.

##### 4.1 Acknowledgements

Edikt is supported by a research development grant from the Scottish Higher Education Funding Council. It is based at the National e-Science Centre in co-operation with EPCC, both at the University of Edinburgh.

#### 5 References

- [1] DAIS Working Group, <http://www.cs.man.ac.uk/grid-db> and <https://forge.gridforum.org/projects/dais-wg>
- [2] N.P. Chue Hong *et al*, *Grid Database Service Specification*, February 16, 2003, [http://www.cs.man.ac.uk/grid-db/papers/DAIS\\_GGF7StatementSpec.pdf](http://www.cs.man.ac.uk/grid-db/papers/DAIS_GGF7StatementSpec.pdf)
- [3] M.A. Antonioletti *et al*, *Grid Database Service Specification*, June 6, 2003, <http://www.cs.man.ac.uk/grid-db/papers/draft-ggf-dais-gdss-ggf8.pdf>
- [4] Sun Microsystems, Inc., *Enterprise Java Beans Technology Downloads and Specifications*, 2003, <http://java.sun.com/products/ejb/docs.html>
- [5] M.P. Atkinson *et al*, *Grid Database Access and Integration: Requirements and Functionalities*, February 17, 2003 [http://www.globalgridforum.org/Meetings/ggf7/drafts/DAIS\\_GGF7RF.pdf](http://www.globalgridforum.org/Meetings/ggf7/drafts/DAIS_GGF7RF.pdf)
- [6] JBoss Organization, *JBoss 2.4 + Documentation*, June 4, 2003, <http://jboss.sourceforge.net/doc-24/>
- [7] Sun Microsystems, Inc., *Sun ONE Application Server 7 Server Architecture Overview*, September 18, 2002, <http://docs.sun.com/source/816-7143-10/srvrarch.html>
- [8] The OGSA-DAI Project, <http://www.ogsa-dai.org.uk/>
- [9] S. Tuecke *et al*, *Open Grid Services Infrastructure (OGSI), DRAFT 0.23*, February 17, 2003, [http://www.globalgridforum.org/Meetings/ggf7/drafts/draft-ggf-ogsi-gridservice-23\\_2003-02-17.pdf](http://www.globalgridforum.org/Meetings/ggf7/drafts/draft-ggf-ogsi-gridservice-23_2003-02-17.pdf)
- [10] S. Tuecke *et al*, *Open Grid Services Infrastructure (OGSI), Version 1.0*, GFD-R-P.15, June 27, 2003, <http://heanet.dl.sourceforge.net/sourceforge/ggf/GFD-R-P.15.pdf>
- [11] John Crupi, Deepak Alur and Dan Malks, *Core J2EE Patterns*, Prentice Hall / Sun Microsystems Press, ISBN:0130648841, <http://java.sun.com/blueprints/corej2eepatterns/>