International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-8 Issue-6, August, 2019

Global Grids and Package Toolkits at 4 Grid Middleware Technologies

P.Meenakshi Sundaram,

Abstract: ***Grid is an infrastructure that involves the integrated and collaborative use of computers, networks, databases and scientific instruments owned and managed by many organizations. Grid applications often involve large amounts of data and/or computer resources that require a secure resource sharing throughout the organization. This makes grid operation and deployment a complex undertaking. Grid middleware provides users with seamless computer skills and uniform access to resources in the heterogeneous grid. A number of toolkits and systems have been developed, most of which are the result of academic research projects worldwide. This chapter focuses on four of these intermediaries: UNICORE, Globus, Legion and Grid bus. It also presents our implementation of a UNICORE resource broker because it did not support this functionality. A comparison of these systems is included based on the architecture, the implementation model and a number of other functions.***

Keywords: Grid, UNICORE, Globus, Middleware.

I. INTRODUCTION

Commodity computers (PCs) and network performance have increased considerably in the last decade, mainly due to faster hardware and sophisticated software. These commodity technologies have been used in a number of applications to solve resource-intensive problems [1] to develop low-cost, high-performance computer systems, commonly referred to as clusters. There are, however, many problems in the fields of science, engineering and business that the current generation of high performance computers cannot track. These problems are often resource-intensive (computational and data) due to their size and complexity and need to work with distributed interdisciplinary application models and components.

Therefore, these applications require a variety of resources not found in a single organization. The ubiquity of the Internet and the web, together with the availability of powerful computers and high-speed broad-based networking technologies as low-cost commodity components, are rapidly changing the computer landscape and society. These technological opportunities have led to the use of large-scale distributed resources to solve large-scale problems, leading to what is commonly known as grid computing [2]. The term "grid" is chosen as an analogy to the electricity grid, providing consistent, comprehensive, reliable and transparent access to electricity regardless of source.

The analogy between computer and electrical grids is discussed in [3]. Grids allow heterogeneous resources, such as computers, databases, visualization devices and scientific instruments, to be shared, exchanged, discovered, selected and aggregated geographically/internet-wide. They have therefore been proposed as the next generation computing problem and global cyber infrastructure to solve major problems in science, engineering and business. In contrast to traditional parallel and distributed systems, grids address issues such as security, uniform access, dynamic discovery, dynamic aggregation and service quality. A number of prototype applications have been developed and experiments in grid planning [4]-[8] have been carried out.

The results of these efforts show that the grid computing paradigm is very promising. In addition, Grids have the potential to share scientific instruments such as particle accelerator (CERN Large Hadron Collider [9]), Australian radio telescope [10] and synchrotron [11], which have been commissioned as national / international infrastructure due to the high cost of ownership and to support on-demand and real-time data processing and analysis. Such a capability will radically increase the opportunities for scientific and technological research and innovation, industrial and business management, application software and services and commercial activities, and so on. A high-level overview of activities in a seamless, integrated computer and collaborative grid environment is shown in Figure 1. The end users interact with the grid resource broker, which detects schedules and processes applications on the grid resources.

The Grid middleware systems must solve several challenges arising from the inherent features of the Grid [12] in order to provide users with a seamless computing environment. The heterogeneity in grid environments due to the multiplicity of heterogeneous resources and the wide range of grid technologies is one of the main technologies. Another challenge is the multiple administrative domains and autonomy problems due to the geographically distributed grid resources in multiple administrative domains owned by various organizations. Other challenges include scalability (performance degradation problem with increasing grid size) and dynamically / adaptability (resource failure problem is high). Middleware systems must dynamically tailor their behavior and use the available resources and services efficiently and effectively.

Revised Manuscript Received on October 30, 2019.
* Correspondence Author
Dr.P.Meenakshi Sundaram*, M.C.A., M.Phil., Ph.D., Assistant Professor, Department Of Computer Science, Marudupandiyar College of Arts and Science, Thanjavur, sundaramp994@gmail.com.

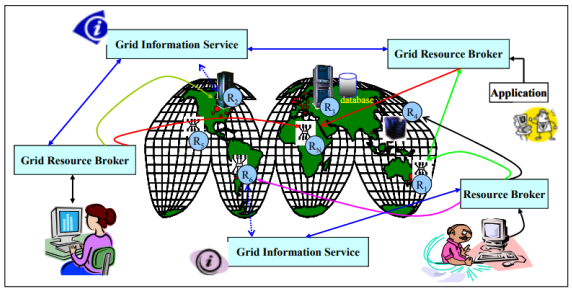


Figure 1: A world-wide Grid computing environment

A great deal of effort has been invested in the design and implementation of middleware software to enable computer grids. Several of these software packages have been deployed successfully and it is now possible to build grids beyond the limits of a single local network. Examples of Grid middleware include UNICORE [13], Globus [13], Legion [16], and Grid bus [17]. These middleware systems aim to provide a grid computing infrastructure where users are connected to computer resources without knowing where the computer cycles are generated. The rest of this chapter provides an insight into the various Grid middleware systems that exist today, followed by a comparison of these systems, and also illuminates the various projects using the above mentioned middleware.

II. OVERVIEW OF GRID MIDDLEWARE SYSTEMS

Figure 2 shows the typical grid architecture of the hardware and software stacks. It consists of four layers: fabric, core middleware, user-level middleware and layers for applications and portals. The grid fabric level comprises distributed resources such as computers, networks, storage devices and scientific tools. The computer resources represent multiple architectures, such as clusters, supercomputers, servers and ordinary PCs running a variety of operating systems (such as UNIX versions or Windows). Scientific tools such as telescopes and sensor networks provide real-time data which can be transmitted directly to computer sites or stored in a database.

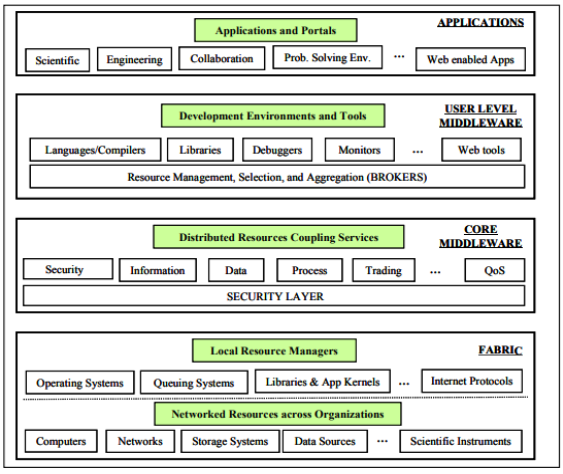


Figure 2: A Layered Grid Architecture and components

Core Grid middleware offers services such as remote process management, resource co-allocation, access to storage, recording and discovering information, security and quality of service (QOS) aspects such as resource reservation and trading. These services abstract the complexity and heterogeneity of the tissue level by providing a consistent approach to accessing distributed resources. The user-level Grid middleware uses the interfaces provided by the low-level middleware to offer abstractions and services at higher levels.

These include application development environments, programming tools and resource brokers for resource management and planning applications for global resource implementation. Grid applications and portals are generally developed using grid-enabled languages and utilities like HPC++ or MPI. An example application, such as parameter simulation or a big challenge problem, requires computer power, access to remote data sets and may need to interact with scientific instruments.

Grid portals offer web-enabled application services where users can submit and collect results through the Web for their jobs on remote resources. In this chapter, the middleware surveyed extends over one or more levels above the grid fabric layer of this generic stack. A brief description for each of them is given in Table 1.

Table 1: Grid middleware systems

Name	Description	Remarks	Website
UNICORE	Vertically integrated Java based Grid computing environment that provides a seamless and secure access to distributed resources.	Project funded by the German Ministry for Education and Research with co-operation between ZAM, Deutscher, etc	http://www.unicore.org
Globus	Open source software toolkit that facilitates construction of computational grids and grid based applications, across corporate, institutional and geographic boundaries without sacrificing local autonomy.	R&D project conducted by the "Globus Alliance" which includes Argonne National Laboratory, Information Sciences Institute and others.	http://www.globus.org
Legion	Vertically integrated Object-based metasystem that helps in combining a large numbers of independently administered heterogeneous hosts, storage systems, databases legacy codes and user objects distributed over wide-area-networks into a single, object-based *metacomputer* that accommodates high degrees of flexibility and site autonomy.	A R&D project at the University of Virginia, USA. The software developed by this project is commercialized through a new company called Avaki.	http://legion.virginia.edu
Gridbus	Open source software toolkit that extensively leverages related software technologies and provides an abstraction layer to hide idiosyncrasies of heterogeneous resources and low-level middleware technologies from application developers. It focuses on realization of utility computing and market-oriented computing models scaling from clusters to grids and to peer-to-peer computing systems.	A research and innovation project led by the University of Melbourne GRIDS Lab with support from the Australian Research Council.	http://www.gridbus.org/

3. UNICORE

UNICORE [13] is a vertically integrated grid computing environment that enables:

- A seamless, secure and intuitive access to resources in a distributed environment-for end users.
- Solid authentication mechanisms included in their management procedures, reduced training effort and support requirements-for grid sites.
- Easy relocation of computer jobs to multiple platforms-both end users and Grid sites. UNICORE follows a three-tier architecture (drawn with ideas from [14]) shown in Figure 3. It consists of a client running on a Java-enabled user workstation or PC, a gateway and multiple Network Job Supervisors (NJS) instances running on dedicated securely configured servers and multiple Target System Interfaces (TSI) running on different nodes that provide interfaces to local resource management systems such as operating systems and batch subsystems. From the point of view of the end user, UNICORE is a client-server system base on a three-tier model:

- User tier: The user runs the UNICORE client at a local workstation or computer.
- Server tier: Each participating computer center defines one or more UNICORE Grid sites (U sites) to which clients can connect.
- Target System level: a website provides access to computing or data resources.

hey are organized as one or more virtual sites (V sites) that can represent the computer center execution and /or storage systems. There are two components in the UNICORE Client interface: JPA (Job Preparation Agent) and JMC (Job Monitor Component). Jobs are built using JPA and employment status and results can be obtained through the JMC. The requests for jobs or status and the results are formulated in an abstract form using the Java classes Abstract Job Object (AJO). The customer connects to a UNICORE portal and submits the jobs through AJOs. The UNICORE Gateway is the single entry point for all UNICORE links to a site. It provides an Internet address and a port that users can use to connect to the SSL gateway.

There are two components in the UNICORE V site: NJS (Network Job Supervisor) and TSI (Target System Interface). The NJS Server manages all UNICORE jobs submitted and authorizes the user by mapping the user certificate to a valid UUDB (UNICORE User Data Base) login. NJS also deals with the embodiment of jobs from the AJO definition into the appropriate concrete command sequences for a specific target execution system based on specifications in the Incarnation Data Base (IDB). UNICORE TSI accepts embodied work components from the NJS and passes them to the local batch execution systems. The features and functions of UNICORE can be summarized as follows:

1 User driven job creation and submission: A graphical interface helps the user create complex and interdependent jobs that can be performed on any UNICORE site without changes in job definition.
2 Job management: The Job management system gives users complete control over jobs and data.
3 Data management: When creating a job, the user can specify which data sets to import or export from U space (set of all files available for a UNICORE job) and which data sets to transfer to another U space. Without user intervention, UNICORE performs all data movement at run time.
4 Application support: As scientists and engineers use specific scientific applications, the user interface is plugged into plug-ins that enables specific application inputs to be prepared.
5 Flow control: A user task can be described as a set of one or more acyclic graphs.
6 Single sign-on: UNICORE offers a single sign-on with certificates X.509V3.
7 Support for legacy jobs: UNICORE supports traditional batch processing by enabling users to include their old job scripts in a UNICORE job.
8 Management of resources: Users select the target system and specify the resources required.

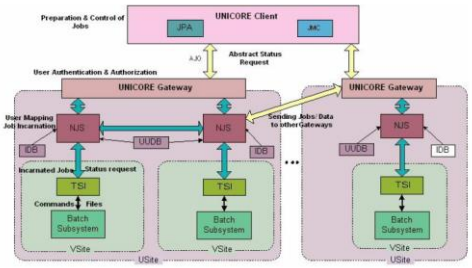


Figure 3: The UNICORE Architecture

The UNICORE client checks the correctness of the work and warns users immediately to correct errors. The main grid tools and application projects using UNICORE as their low-level middleware include: Euro Grid [18] and its applications-Bio Grid [19], Metro Grid and CAE Grid, Grid interoperability Project (GRIP)[20], Open Mol Grid[19] and Japanese NAREGI (National Research Grid Initiative) [22].

OVERVIEW OF JOB CREATION, SUBMISSION AND EXECUTION IN UNICORE MIDDLEWARE

The UNICORE customer helps to create, manipulate and manage complex, interdependent multi-system jobs, multi-site jobs, synchronize jobs and move data between systems, sites and storage spaces. The client creates an Abstract Job Object (AJO) as a serialized Java object or in XML format. The UNICORE (NJS) Server performs

- Incarnation of AJO into targeted system-system actions
- Synchronization of actions (work flow)
- Transfer of jobs and data between User Workstation, Target Systems and other sites

OBSERVATION OF STANDING THE 2 MAIN AREAS OF UNICORE ARE:

1. Seamless specification of work to be done on a remote site and

2. Transmission of specifications, results and related data.

The seamless UNICORE specification is covered by a collection of Java classes loosely known as the AJO(Abstract Job Object) and the transmission is defined in the UNICORE Protocol Layer(UPL).The UPL is a protocol which transmits data independently of its form. The UPL classes are included in the package org.unicore.upl and the package org.unicore.upl with some auxiliary functions of the package org.unicore.utility. Org.unicore.ajo, org.unicore.outcome, org.unicore.resources and org.unicore.idiomatic are the main packages for AJO. The AJO can specify a UNICORE client application such as a Job Preparation Agent (JPA) or Job Monitor Controller (JMC) to a UNICORE server (NJS).

One objective of the AJO is to allow seamless specification of jobs that can be redirected to different computer sites only by changing the address, the specification of the work remains the same-regardless of different site policies, different executables and options, different licensing policies, etc. The mapping of the seamless specification to the actual site values, known as the embodiment in UNICORE, is carried out by the NJS at the target site at the current time. The execution atom at UNICORE is an abstract task. NJSs execute user's abstract Jobs. An abstract job includes one or more subtasks.

The subtasks are abstract actions and define simple actions like fetching, compiling, executing files. Abstract jobs are also abstract actions and can therefore be contained in a parent's abstract job. When an NJS begins to perform an abstract job, it creates a directory for the abstract job on a file system. This directory is known as the Job's abstract space. All files used by the child are supposed to be in the space. Most abstract actions have no direct access to the file system of a website (X space). Any file used by Abstract Actions must be imported into the U space before it is used and any files that must be saved from the U space explicitly. The file can be saved by exporting it, returning it with the results of the AJO or spreading it to a semi-permanent holding area.

The U space is deleted when the AJO is finished. The child's abstract actions of an abstract job are contained in a Directed Acyclic Graph (DAG) that defines the execution order. Successor abstract actions start implementation when all their predecessor abstract actions have successfully completed execution. If an abstract action fails, none of its successor abstract actions will be carried out. The results of the performance of abstract actions are known as results. The results are the stout and steer produced by an executable, logging and status information from the NJS and/or the results of the abstract action. Each kind of abstract action corresponds to an outcome type.

The sub classes of org.unicore.ajo Abstract Task are the main classes for the definition of work. These define the work carried out by the target system outside the NJS and thus define tasks such as executing, linking, compiling and manipulating various files. These tasks can require resources such as number of processors, quantity of memory, time and any external software packages. A client can ask the website for a description of the resources supported and therefore only request available resources. The other major grouping of abstract actions is based on those capable of manipulating other abstract actions, for example. Return current status, return results, kill, etc. The following is the UNICORE object hierarchy:

- Abstract Action: Any UNICORE actions parent class.
- Action Group: UNICORE actions container.
- Abstract Job: a remote Action Group that can run.
- Repeat Group: loop actions.
- Task Abstract: computer action, e.g. Filecopy.
- Abstract Service: action for service, e.g. Kill work.
- Conditional Action: action if-then-else.

Action Group contains a DAG of Abstract Actions that shows dependencies between actions (nodes) that define the flow of control. Actions in the DAG can be any Abstract Action subtype. An action begins if all these predecessors are "DONE". For control, the following subclasses of "DONE" are used.

- SUCCESSFUL: the abstract action without error completed.
- NOT SUCCESSFUL: Failed an abstract action.
- NEVER RUN: Failed by the Abstract Action predecessor.
- NEVER TAKEN: The abstract is in a non-conditional action branch. The workflow structures in UNICORE allow.
- Automation of complex multi-site, multi-system Jobs.
- Run computer experiments such as parameter studies.
- Use all UNICORE features, such as safety and seamless.

4. GLOBUS

The Globus project provides an open source software toolkit [13] to build grids and grid-based applications. It allows computer power, databases and other tools to be shared securely online across corporate, institutional and geographical boundaries without sacrificing local autonomy. The core services, interfaces and protocols in the Globus toolkit allow users to access remote resources seamlessly while maintaining local control over who and when can use resources.

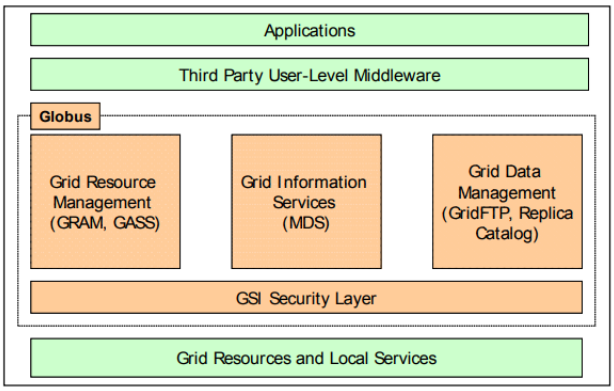


Figure 4: The Globus Architecture

The Globus architecture shown in Figure 4 contains three main service groups that are accessible via a security layer. These groups are management of resources, data management and information services. The local service layer includes operating system services, network services such as TCP/IP, cluster scheduling services provided by Load Leveler, job submissions, queue queries, and so on. The higher layers of the Globus model allow several or heterogeneous clusters to be integrated. The core service layer contains the building blocks of the Globus toolkit for security, job submission, data management and resources information management. The high level of services and tools include tools that integrate lower level services or implement missing features.

5. LEGION

Legion [16] is a middleware system that combines a very large number of independently managed heterogeneous hosts, storage systems, legacy databases and user objects distributed across wide networks into a single coherent computing platform. Figure 5 shows Legion'smiddleware architecture.

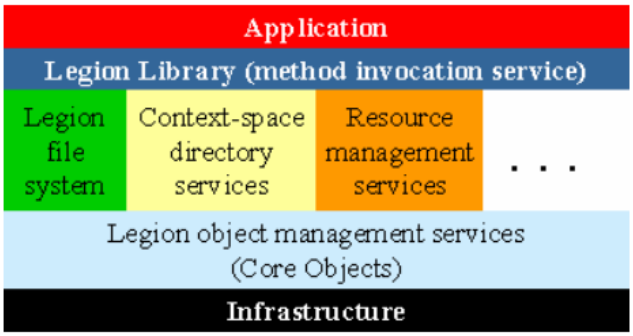


Figure 5: Legion Architecture.

It is structured as a distributed "object" system-active processes that communicate using a uniform remote invocation service.

All hardware and software resources in a grid system will be represented by Legion objects. Legion's fundamental object models are described using an interface description language (IDL), and are compiled and linked to implementations in a given language.

This approach enables component interoperability between multiple programming languages and heterogeneous execution platforms. Since all elements of the system are objects, they can communicate with each other regardless of location, heterogeneity or details of implementation, addressing encapsulation and interoperability or details of implementation, addressing encapsulation and interoperability problems. A "class object" is used for the definition and management of its Legion object. Class objects are responsible at the system level; they control the creation of new instances, schedule the execution, activate and deactivate instances and provide information about their current location to client objects wishing to communicate with the instances. In other words, classes function as system managers and policy makers. Meta classes are used to describe an instance of classes.

The main grid tools and test beds used by Legion as their low-level middleware are: NPACI Test bed [42], Nimrod-L [41] and NC Bio Grid [40]. Furthermore, it was used in the study of axially symmetrical steady flow [39] and protein folding [38].

6. GRIDBUS

The Grid bus Project [17] is a multi-institutional open-source project led by the GRIDS Lab at Melbourne University. It develops service-oriented cluster and grid middleware technologies for e Science and e Business applications. It uses related software technologies extensively and provides an abstraction layer to hide the idiosyncrasies heterogeneous resources and low-level middleware technologies from application developers. It also focuses extensively on the implementation of the scaling of utility computer models from clusters to grids and peer-to-peer computer systems. It employs economic models [43] to manage shared resources efficiently and promotes the commoditization of its services. It therefore improves the tradability of grid services and manages the supply and demand of resources efficiently. Grid bus promotes the commercialization of grid services at different levels:

- Raw resource level (e.g. selling CPU cycles and storage resources)
- Application level (e.g. drug design molecular docking operations [7])
- Aggregate services (e.g. multi-domain brokering and re-selling services)

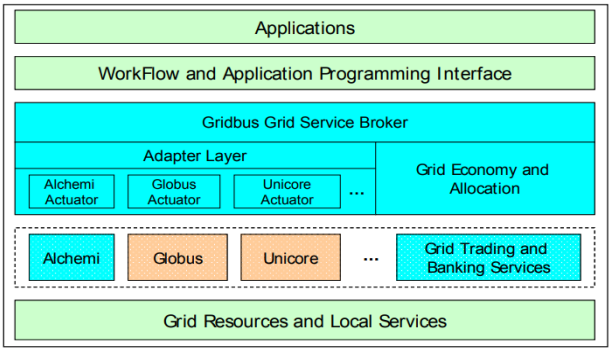
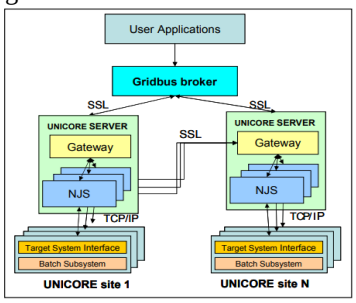


Figure 6: The Grid bus Architecture

7. Implentation Of Unicore Adapter Of Grid Bus

Broker Legion provides its own grid broker as a complete integrated system that facilitates the management, selection and aggregation of resources and the scheduling of applications for global resources while Globus and UNICORE do not provide their own. Globus does, however, have a number of third-party grid broker implementations such as Nimrod-G, Grid bus broker and Condor-G, but UNICORE has no such add-on service. The lack of brokerage services in UNICORE motivates the need to transfer a grid broker to UNICORE.

This section describes the extension of the Grid bus broker and the implementation of the adapter for the UNICORE middleware. The grid bus scheduler has a platform-independent view of the nodes and only focuses on their performance. The broker can therefore operate through different middleware. The Grid bus is already able to use Globus and Alchemi middleware on resources that are enabled by grid. The Grid bus broker can also be designed to work with UNICORE middleware by extending the Grid bus Classes of Computer Server, Job Wrapper, Job Monitor and Job Output. The main components of the grid bus broker implementation on UNICORE are shown in the UML diagram in Figure 7.



0

Figure 7: Schematic for interfacing Grid bus broker on UNICORE middleware

UNICORE COMPUTESERVER

The object Computer Server in the Grid bus broker describes nodes in a system. The attributes of the nodes enabled by UNICORE can be described in UNICORE Computer Server by extending the Computer Server class for UNICORE. To create access to resources, we must first create an identity used for the execution of AJO.

Identity identity = new Identity (new File (),);

The created identity is then used to establish an SSL connection to the gateway and a link to the V site.

Reference reference = new Reference.SSL (, identity);

V siteTh vsite = new V siteTh (reference,);

As the information is extracted directly from the command line, the gateway does not need to check. Dynamic applications can only accept certain gateway addresses and subsequently present the user with the list of ports and sites. Resources can also be obtained from the V sites using the V site Manager, which offers a global view of all known V sites.

Unicore Job Wrapper

The main function of this object is to convert the job specified by the Grid bus broker to a format that can be understood by the UNICORE middleware computer server using the XML-based sweep parameter (xPSS).

The xPSS document contains three major command types: SUBSTITUTE, COPY and EXECUTE. We need to copy the required AJO files from the client machine to the remote machine for the COPY command and copy the result files back to the local machine. In the first case, the required files are first placed in the Incarnate Files, which generates files from the data in the instance in the U space. It is an alternative way to transfer files to U spaces. In this case, the bytes of the files are placed in the AJO and this method is therefore only useful in small files. Large files ought to be streamed. Finally, files are placed in a portfolio that is transferred together with the AJO to be executed.

Method 1:

```
Make Portfolio mp1=new Make Portfolio ();
File file=new File ();
File Input Stream fin=new File Input Stream (file);
byte [] file contents=new byte[(into) file. Length()];
fin. Read (file contents);
Incarnate Files inf1=new Incarnate Files ("Required Files");
Inf1.addFile (.file contents); mp1.addFile ();
```

Method 2:

```
Portfolio port folio = new Port folio ();
Port folio Th files = new Port folio Th(portfolio,);
```

In the second case, the result files in the remote machine are added to the job Outcome which is transferred to the local machine. This is because all U space files are deleted at the end of execution of AJO.

Step 1: Let the NJS know about the files (make a Portfolio)

```
Make Portfolio mp2 = new Make Portfolio ()
Mp2.addFile (<Result file name in U space>);
```

Step 2: Save files

```
Copy Portfolio to Outcome cpto = new Copy PortfolioToOutcome ();
Cpto.setTarget (mp2.getPortfolio ().getId ());
```

For the EXECUTE command, the executable in the plan file is converted into a script and an Execute Script Task is created which is included in the AJO.

```
String the script = "date\n"
The script+= "hostname\n"
inf.addFile ("script", the_script.getBytes ());
Make Portfolio mp = new Make Portfolio ("AJO Example");
mp.addFIle ("script");
Execute Script Task est. = new Execute Script Task ("AJO Example Task");
Est. set Script Type (ScriptType.CSH);
est.setExecutable (mp.getPortfolio ());
```

In the Arcon client library, the AJO is created by adding the components and their dependencies and submitted to the remote machine using the job manager. Consign Synchronous method in com.fujitsu.arcon.servlet.JobManager.

```
Abstract Job ajo = new Abstract Job ("AJO Example");
ajo.addDependency (inf.mp);
ajo.addDependency (inf1.mp1);
ajo.addDependency (mp.est);
ajo.addDependency (est.mp2);
ajo.addDependency (mp2.cpto);
Outcome = JobManager.consignSynchronous (ajo. Vsite);
```

The Unicode Job Wrapper also starts the Unicode Job Monitor thread which is responsible for monitoring the status of the job execution.

UNICORE JOB MONITOR

The Unicode Job Monitor is used to monitor job performance and to display detailed status information during performance. The status information includes all abstract and abstract tasks, such as Make Portfolio and ExecuteScriptTask. In the event of job failure or job completion, it is also responsible for terminating the job.

UNICORE JOB OUTPUT

Unicode Job Output is used to recover the results of the job as its name suggests. It shows the content of the standard output and the standard error files on the local machine from the Computer Server. When other result files are produced as a result of job execution, Unicode Job Output renames these files to the format "filename.jobid" and saves them in the local folder. The result is obtained by using the Job Manager.getOutcome function of the Job Manager from the client side. All the result files are therefore written to local storage when the AJO is completed successfully.

```
Collection c = outcome.getFilesMapping ().get (cpto. GetId ());
        Iterator I = c.iterator ();
        While (i.hasNext ())
        {
        File f = (File) i. Next ();
        System.out.println ("\n\result FILE:"+f.getCanonicalPath () +"\nCONTENTS\n");
        Buffered Reader reader = new Buffe red Reader (new File Reader (f));
        String line;
        While ((line = reader.readLine ())!= null){System.out.println(line);}
```

COMPARISON OF MIDDLEWARE SYSTEMS

Figure 8 compares the middleware surveyed according to the services provided throughout the grid architecture stack. UNICORE and Legion are integrated vertically and closely linked. Both server and client components are supplied. Globus follows a "service bag" and offers a wide range of basic tools that can be selectively used to build grid systems. For example, although Grid bus has its own low-level independent middleware, its broker is designed to work with Globus. Although spread over the stack, the Grid bus component set is not as tightly integrated as UNICORE and Legion and can be used completely independently of each other.

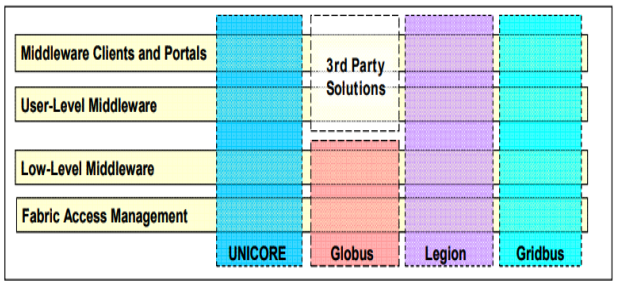


Figure 8: Comparison of UNICORE, Globus, Legion and Grid bus

Table 2 provides a comparison of the different middleware systems based on their architecture, model and category. A further comparison can be found in [37] between Globus and Legion.

Table 2: Comparison of Grid Middleware Systems

Middleware / Property	UNICORE	Globus	Legion	Gridbus
Focus	High level Programming models	Low level services	High level Programming models	Abstractions and market models
Category	Mainly uniform job submission and monitoring	Generic computational	Generic computational	Generic computational
Architecture	Vertical multi tiered system	Layered and modular toolkit	Vertically integrated system	Layered component and utility model
Implementation Model	Abstract Job Object	Hourglass model at system level	Object-oriented metasystem	Hourglass model at user level
Implementation Technologies	Java	C and Java	C++	C, Java, C# and Perl
Runtime Platform	Unix	Unix	Unix	Unix and Windows with .NET (Alchemi)
Programming Environment	Workflow environment	Replacement libraries for Unix & C libraries. Special MPI library (MPICH –G), CoG (Commodity Grid) kits in Java, Python, CORBA, Matlab, Java Server Pages, Perl and Web Services	Legion Application Programming Interfaces (API). Command line utilities	Broker Java API XML-based parameter-sweep language Grid Thread model via Alchemi.
Distribution Model	Open source	Open source	Not open source. Commercial version available	Open source
Some Users and Applications	EuroGrid [18], Grid Interoperability Project (GRIP) [20], OpenMolGrid [19], and Japanese NAREGI [22].	AppLeS [28], Ninf [30], Nimrod-G [29], NASA IPG [36], Condor-G [31], Gridbus Broker [32], UK eScience Project [33], GriPhyN [35], and EU Data Grid [34].	NPACI Testbed [42], Nimrod-L [41], and NCBioGrid [40]. Additionally, it has been used in the study of axially symmetric steady flow [39] and protein folding [38] applications.	ePhysics Portal [52], Belle Analysis Data Grid[50], NeuroGrid [48], Natural Language Engineering [53], HydroGrid [46], and Amsterdam Private Grid [47].

III. CONCLUSION

There has been extensive research into the development of Grid middleware systems and most of these systems have been successfully implemented in other grid projects and applications. The Globus toolkit is one of today's low-level grid middleware. It offers key services such as access to resources, data and security infrastructure. UNICORE is a Java-based grid computing system used in projects such as EUROGRID and GRIP. The Grid bus toolkit makes extensive use of related software technologies and provides an abstraction layer to hide idiosyncrasies of heterogeneous resources and low-level middleware technologies from application developers. It focuses on the realization of the model of utility computing from clusters to grids and computer systems. Legion is a middleware system based on objects implemented in projects such as the Boeing R&D project. Comparing the grid middleware systems shows that their functionality is considerably overlapped. The main difference was found in the architecture or model of the implementation. The examination of the features of UNICORE shows that it does not provide brokerage services that manage or schedule calculations across the world. This deficiency led us to port the resource broker Grid bus to the UNICORE environment.

REFERENCES

1. R. Buyya (editor), High Performance Cluster Computing, Prentice Hall, USA, 1999.
2. I. Foster and C. Kesselman (editors), The Grid: Blueprint for a Future Computing Infrastructure, Morgan Kaufmann Publishers, USA, 1999.
3. M. Chetty and R. Buyya, Weaving Computational Grids: How Analogous Are They with Electrical Grids ?, Computing in Science and Engineering (CiSE), ISSN 1521-9615, Volume 4, Issue 4, Pages: 61-71, IEEE Computer Society Press and American Institute of Physics, USA, July-August 2002.
4. M. D. Brown et al., The International Grid (iGrid): Empowering Global Research Community Networking Using High Performance International Internet Services, April 1999, http://www.globus.org/research/papers.html
5. S. Smallen et al., Combining Workstations and Supercomputers to Support Grid Applications: The Parallel Tomography Experience, 9 th Heterogenous Computing Workshop (HCW 2000, IPDPS), Mexico, April, 2000
6. SC 2000 Net Challenge, http://www.npaci.edu/sc2000/netchallenge.html
7. R. Buyya, K. Branson, J/ Giddy, and D. Abramson, The Virtual Laboratory: A Toolset for Utilising the WorldWide Grid to Design Drugs, 2nd IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2002), 21 - 24 May 2002, Berlin, Germany.
8. iGrid 2002, International Exhibition on Grid Applications, http://www.igrid2002.org
9. CERN LHC Computing Grid Project, http://lhcgrid.web.cern.ch/LHCgrid/
10. Australia Telescope National Facility, http://www.atnf.csiro.au/
11. Australian Synchrotron Project, http://www.synchrotron.vic.gov.au
12. M. Baker, R. Buyya, and D. Laforenza, Grids and Grid Technologies for Wide-Area Distributed Computing, International Journal of Software: Practice and Experience (SPE), Volume 32, Issue 15, Pages: 1437-1466, Wiley Press, USA, December 2002.
13. J. Almond, D. Snelling, UNICORE: Uniform Access to Supercomputing as an Element of Electronic Commerce. Future Generation Computer Systems 613(1999), 1-10
14. M. Romberg, The UNICORE architecture: seamless access to distributed resources, Proceedings of The Eighth International Symposium on High Performance Distributed Computing, Redondo Beach, CA, USA. 1999
15. I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit", International Journal of Supercomputer Applications, 11(2): 115-128, 1997.
16. A. Grimshaw, W. Wulf, The Legion vision of a worldwide virtual computer. Communications of the ACM 1997; 40(1)
17. R. Buyya and S. Venugopal, The Gridbus Toolkit for Service Oriented Grid and Utility Computing: An Overview and Status Report, Proceedings of the First IEEE International Workshop on Grid Economics and Business Models (GECON 2004, April 23, 2004, Seoul, Korea), 19-36pp, ISBN 0-7803-8525-X, IEEE Press, New Jersey, USA
18. The EuroGrid Project - http://www.eurogrid.org/
19. P. Bala, J. Pytlinski, M. Nazaruk, BioGRID-An European grid for molecular biology, Proceedings of 11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11), Edinburgh, UK, 2002.
20. R. Menday, P. Wieder, GRIP: The Evolution of UNICORE towards a Service-Oriented Grid, Cracow Grid Workshop, October 27 - 29, 2003, Cracow Poland.
21. Open Computing Grid for Molecular Science and Engineering (OpenMolGRID) - http://www.openmolgrid.org/
22. NAREGI Project - http://www.naregi.org/
23. I. Foster, C. Kesselman, G. Tsudik, S. Tuecke, A Security Architecture for Computational Grids. Proc. 5th ACM Conference on Computer and Communications Security Conference, San Francisco, CA, USA.
24. K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, S. Tuecke. A Resource Management Architecture for Metacomputing Systems. Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing, Orlando, FL, USA, 1998
25. S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, S. Tuecke. A Directory Service for Configuring High-Performance Distributed Computations. Proc. 6th IEEE Symposium on High-Performance Distributed Computing, Portland, OR, US, 1997
26. B. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, S. Tuecke Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing. IEEE Mass Storage Conference, San Diego, CA, USA, 2001.
27. J. Bester, I. Foster, C. Kesselman, J. Tedesco, S. Tuecke. GASS: A Data Movement and Access Service for Wide Area Computing Systems. Sixth Workshop on I/O in Parallel and Distributed Systems, Atlanta, GA, USA, May 5, 1999.
28. F. Berman and R. Wolski. The AppLeS Project: A Status Report. 8th NEC Research Symposium, Berlin, Germany, May 1997
29. D. Abramson, J. Giddy, and L. Kotler, High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid?, Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS 2000), May 1-5, 2000, Cancun, Mexico, IEEE CS Press, USA, 2000.
30. M.Sato, H. Nakada, S. Sekiguchi, S. Matsuoka, U. Nagashima and H. Takagi, Ninf: A Network Based Information Library for Global World-Wide Computing Infrastructure, Proceedings of the International Conference on High Performance Computing and Networking Europe (HPCN Europe), Vienna, Austria, April 28-30, 1997.

31. J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke, Condor-G: A Computation Management Agent for Multi-Institutional Grids, Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC10) San Francisco, California, August 7-9, 2001.
32. S. Venugopal, R. Buyya and L. Winton, A Grid Service Broker for Scheduling Distributed Data-Oriented Applications on Global Grids, Technical Report, GRIDS-TR-2004-1, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia, February 2004.
33. T. Hey and A. E. Trefethen, The UK e-Science Core Programme and the Grid, Future Generation Computer Systems, Volume 18, Issue 8, October 2002, Pages 1017-1031.
34. W. Hoschek , J. Jaen-Martinez, A. Samar , H. Stockinger , K. Stockinger, Data Management in an International Data Grid Project, Proceedings of the first IEEE/ACM International Workshop on Grid Computing, (Springer Verlag Press, Germany), India, 2000.
35. Grid Physics Network (GriPhyN).http://www.griphyn.org/ (Accessed Jun 2004)
36. W. Johnston, D. Gannon, and B. Nitzberg. Grids as production computing environments: The engineering aspects of NASA's Information Power Grid. In Proc. Eighth IEEE International Symposium on High Performance Distributed Computing, Redondo Beach, CA, USA, 1999
37. M. Baker and G. Fox, Metacomputing: Harnessing Informal Supercomputers, In [1], Prentice Hall, New Jersey, USA, 1999.
38. A. Natrajan, M. Crowley, N. Wilkins-Diehr, M. Humphrey, A. D. Fox, A. Grimshaw, C. L. Brooks III, Studying Protein Folding on the Grid: Experiences using CHARMM on NPACI Resources under Legion, Proceedings of the 10 th International Symposium on High Performance Distributed Computing (HPDC-10), August 7-9, 2001
39. N. Beekwilder, A. Grimshaw, Parallelization of an Axially Symmetric Steady Flow Program, CS Technical Report CS-98-10, University of Virginia, USA, May 29, 1998.
40. NC-BioGrid - http://www.ncbiogrid.org/
41. D. Abramson, Nimrod on Legion Middleware, http://www.dstc.edu.au/Research/activesheets-ov.html
42. A. Grimshaw, A Worldwide Virtual Computer for an Advancing Legion of Applications, NPACI, http://www.npaci.edu/enVision/v15.2/legion.html
43. R. Buyya, Economic-based Distributed Resource Management and Scheduling for Grid Computing, PhD Thesis, Monash University, Melbourne, Australia, April 12, 2002.
44. A. Luther, R. Buyya, R. Ranjan, and S. Venugopal, "Alchemi: A .NET-based Grid Computing Framework and its Integration into Global Grids", http://www.gridbus.org/papers/Alchemi.pdf
45. R. Buyya, D. Abramson, and J. Giddy, An Economy Driven Resource Management Architecture for Global Computational Power Grids, The 2000 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2000), Las Vegas, USA, June 26-29, 2000.
46. J. Rahman and G. Davis, TIME and DIME: Supporting Natural Resource Modelling with .NET and Alchemi, CSIRO, http://alchemi.net/projects.html
47. A. Warmenhoven, Amsterdam PrivateGrid, The Netherlands, http://www.private-grid.nl/
48. R. Buyya, S. Date, Y. Mizuno-Matsumoto, S. Venugopal, and D. Abramson, Composition of Distributed Brain Activity Analysis and its On-Demand Deployment on Global Grids, New Frontiers in High-Performance Computing: Proceedings of the 10th International Conference on High Performance Computing (HiPC 2003) Workshops (Dec. 17, 2003, Hyderabad, India), ISBN: 81-88901-05-9, Elite Publishing House, New Delhi, India.
49. R. Buyya et. al, Global Data Intensive Grid Collaboration, HPC Challenge Proposal and Demonstration, IEEE/ACM Supercomputing Conference (SC 2003), Nov. 2003 – http://www.gridbus.org/sc2003/
50. BADG Project, Belle Analysis Data Grid, http://epp.ph.unimelb.edu.au/epp/grid/badg/
51. Australian Virtual Observatory (Aus-VO) - http://www.aus-vo.org/
52. B. Beeson, S. Melnikoff, S. Venugopal, and D. Barnes, A Portal for Grid-enabled Physics, Australian Virtual Observatory, http://www.aus-vo.org/.
53. B. Hughes, S. Venugopal, R. Buyya, Grid-based Indexing of a Newswire Corpus, Technical Report, GRIDSTR-2004-4, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia, June/July 2004.
54. I. Foster, C. Kesselman, and S. Tuecke, The anatomy of the grid: Enabling scalable virtual organizations, International Journal of High Performance Computing Applications, vol. 15, pp. 200-222, Sage Publishers, London, UK, 2001.