A Unicore Globus Interoperability Layer

(Draft of a Use Case Study for GPA WG

This draft is subject to review and revision)

Dr. David Snelling & Dr. Sven van den Berghe[*],
Dr. Gregor von Laszewski[†††]
Philipp Wieder[***], Dr. Jon MacLaren and Dr John Brooke[†],
Dr. Denis Nicole[**], Hans-Christian Hoppe[††]

(on behalf of the GRIP EU Project IST-2001-32257)

Abstract

We approach the issue of defining the set of minimal Grid services by considering the problem of interoperability of different Grid systems. On the one hand if we have identified the minimal set of Grid services then any well-composed resource request expressed in terms of this minimal set will be honoured by other Grids even if some translation may need to be made between different resource description mechanisms. On the other, if we are seeking to identify such a minimal set then looking at practical issues of interoperability can help since the minimal set can be defined as the intersection of all interoperability sets between objects that are considered to be Grids. This is a constructionist approach and avoids a precise definition of what a Grid is by refining from systems commonly agreed to be Grids. This case study uses two well-established systems, Globus and UNICORE, that implement reasonably complete solutions for the metacomputing Grid problem. An additional benefit of our approach, providing interoperability between Globus and UNICORE, is that it would result in an advanced set of Grid services that gain strength from each other, addressing the problem of what is a usable as opposed to minimal Grid. Here we gain pointers to augmenting each system to utilise features not initially contained in the other, i.e. beyond the minimal set. This can be considered as the union (in the set-theoretic sense) as opposed to

[*] Fujitsu European Laboratories., {snelling,svdb}@fle.fujitsu.com
[†††] Argonne National Laboratory, gregor@mcs.anl.gov
[***] Forschungszentrum Juelich GmbH, ph.wieder@fz-juelich.de
[†] University of Manchester, {jon.maclaren,john.brooke}@man.ac.uk
[**] University of Southampton, dan@ecs.soton.ac.uk
[††] Pallas GmbH, hoppe@pallas.com

intersection of two Grid implementations. This paper outlines some of these parallels and differences as they relate to the development of an interoperability layer between Unicore and Globus. Given the increasing ubiquity of Globus, what emerges is the desire for a hybridised facility that utilises the Unicore work-flow management of complex, multi-site tasks, but that can run on either Unicore- or Globus-enabled resources. The technical challenge in achieving this, addressed in this paper, consists of mapping resource descriptions from both grid environments to an abstract format appropriate to work-flow preparation, and then the instantiation of work-flow tasks on the target systems. Other issues such as reconciling disparate security models are addressed, authentication being considered as part of the minimal set.

1. The Grid Interoperability Problem

This Use Case paper addresses the current concern of the GPA WG in identifying a minimal set of Grid services. It proposes to use the interoperability issue to shed light on this question, since two Grids must have a common set of services via which they interoperate (although these may be described in different languages in either Grid). A possible way of approaching the minimal services problem is to define the minimal set as the intersection of all such common sets of interoperable common sets, provided that the intersection is extended to give it "nice" properties of closure (e.g. services are consistent and complete in the sense that appeal does not have to be made outside the set for providing the defined minimal functionality). Even if this concept turns out to be a little too idealistic, in a practical sense it can be seen that identifying services which are used consistently by a number of differing Grid systems is an indication that they are somehow capturing the essentials of what a Grid is. We note that these considerations lead very directly to considering the ontology problem for Grid services, solving this problem is one of the very important steps to a true Semantic Grid.

The very dynamic nature of Grid computing has meant that a number of different definitions of a computational Grid have been given. This is not surprising, for comparison in the field of nonlinear systems there a number of different usages of the term "chaos" which are not completely compatible even though there is a deep definition of a chaotic attractor which is mathematically precise[1]. In [Fost98] reference was made to the electrical Power Grid model and in [Fost01] Grids were envisaged as dynamically constituted Virtual Organizations, leading to the Grid Services concept presented in [Fost02] and known as OGSA. The VO concept is interesting in that a very similar concept was proposed quite independently and termed a miniGrid [Broo00]. OGSA has the potential of reconciling some of the differing Grid concepts in the manner that the precise mathematical definition of "chaos" unifies the looser, more case-specific, usages widespread in mathematics and physics literature. For our study the Grid Services and Virtual Organization viewpoints are particularly apt in that they focus attention on the concept of the joining of organizations and groups of services, which is where interoperability is highlighted.

For the purposes of our study we wish to highlight another important aspect common to most usages of Grid. This is the expectation that there exist two important groupings used within the VO concept. One is the group of Resource Requestors (RR) and the other the group of Resource Providers (RP). In the metacomputing usage of the Grid a resource requestor would normally be a users "job", i.e. a request for various processing, storage and transfer operations associated with some well defined task usually initiated from a single point (e.g. a specific workstation). This is a one-to-many mapping from the space of RR to the space of RP. In more sophisticated scenarios the RP space can recursively

[1] A topologically transitive set containing a dense set of periodic orbits. These two conditions imply the often quoted "sensitivity to initial conditions" which latter is not sufficient to describe chaotic attractors.

cast itself as an object in the RR space by passing on the resource request (onward transfer of service requests). The Globus and UNICORE Grid middleware systems that we examine in this case study have a particular orientation to this mode of Grid usage. We note that the human usage of the Grid comes in here as an "invisible" factor. The one-to-many concept comes from a single person submitting a complex request. In the case of collaborative working, however, there is a many-to-many model and the interactions of the RR and RP spaces are highly dynamic. The Grid abstraction is particularly useful for examining such complex usage patterns since it allows each physical resource in the Grid to be used in either an RR or RP context. It is in this sense that one can genuinely consider the Access Grid [Agdoc1] as a Grid in which each AG node is simultaneously and persistently in both the RR and RP spaces (full many-to-many interaction). The RR and RP spaces come naturally from the consideration of the implications of resource sharing in a Virtual Organization. We use the work "spaces" loosely, whether these can be considered as mathematically well-defined spaces (e.g. vector spaces) needs to be investigated.

These considerations are interesting and worthy of future study. However for this current Use Case paper we revert to the more traditional (i.e. over 10 years old) metacomputing usage of Grid. Much current work can be cast into this framework (for example most of the various computational and data Grids) and it is to this area that the two interoperable systems Globus and UNICORE are primarily directed in their current versions. We will not fully address the newer OGSA framework although it has been shown elsewhere that the Grid middleware described here can be adapted to use OGSA [Snell02].

Globus has been described as a Toolkit for the Grid and Unicore as a grid enabled work-flow environment. This distinction probably best describes the differences in approach. Globus provides a collection of tools and an architecture that constitute an infrastructure for Grid enabled application development. Unicore on the other hand provides a vertical solution for performing work-flow task management across multiple disparate resources. Unicore is engineered to allow existing applications and their users to become mobile [Erwi01]. This mobility requires management of computational and data resources and their coordination. The Unicore work-flow services (see below) support the management of the complex, multi-site tasks that arise in this mobile environment.

In addition to EUROGRID [Euro02], the European Commission has recently funded the Grid Interoperability Project,[2] to investigate the issues arising from interoperability between Grids. This paper outlines some of the parallels and differences as they relate to the development of an interoperability layer between Unicore and Globus. Given the established and desirable high-level workflow GUI that Unicore provides, and the pervasiveness of the Globus Toolkit on high-end compute resources, it is natural to hybridise the two by fitting the high-level Unicore functionality over the lower-level components of the Globus Toolkit. Our interoperability case study now becomes clearly defined:

[2] GRid Interoperability Project, IST-20001-32257.

"Can a workflow request constructed in the RR space of UNICORE be realised in the RP space of Globus?"

More colloquially, "Can Globus run a UNICORE job?". Since Globus and UNICORE are well-constructed systems that have been used in demanding real environments we know that each can map from RR to RP space within their own resource description mechanisms and can handle issues such as authentication and instantiation of workflow requests.

2. Background

In this section we discuss those aspects of both Globus and Unicore that are relevant to the development of an interoperability layer between the two.

2.1. Unicore

The mechanisms and architecture of Unicore have been presented elsewhere [Erwi97, Unic02]. The reference implementation (Open Source available [Unic02]) is written in Java. Here we highlight some of those features that provide a work-flow environment for application end-users.

The Unicore resource model, contained in the Incarnation Data Base (IDB), specifies the usual resources of a computer facility such as processing, memory, and disk, but also software resources available at the site. These resources are stored and managed as Java objects and described in a simple text language. In addition to providing specifications of the resources, the IDB includes re-write rules for how abstractions of resources and tasks in the Abstract Job Object (AJO) are to be "incarnated" into their system and site specific forms.

Using the information in the resource set for each site, the user constructs a work-flow style description of the tasks to be performed on a collection of sites. This description is in the form of an AJO. The AJO, as its name implies, is abstract and does not include site or system specific information, e.g. paths to executables or flags for compilers, etc. Once an AJO has been constructed, its contents are signed by the user using her X.509 certificate and *consigned* to a Unicore site (Usite) for processing. The target Vsite within this Usite is called the *primary* Vsite.[3]

At the primary Vsite, AJO processing includes user mapping, authentication, and authorization; task incarnation (for tasks, including file transfer, to be performed at this Vsite); and consignment of sub-AJOs to other Vsites at this or other Usites. Note that secondary Usites do not need to trust the primary Usite, each sub-AJO was signed by the user prior consignment to the primary Vsite.

[3] A Unicore site (Usite) has one security gateway and may have several Virtual sites (Vsites) with in it.

Complex work-flow structures are possible in this framework, including the following examples:

1. Data pre-processing at site A, simulation at site B, and post-processing at site C.

2. Iterative re-execution of a simulation until a termination criteria is met. The simulation may be a complex job, such as (1) above.

3. Simple application steering, in which a job "holds" itself at some point and waits for the user to check intermediate results before releasing the job.

4. More complex application steering, in which jobs are monitored and steered during execution. This may include visualization.

5. Conditional execution of tasks, or whole sub jobs, depending on the results of other tasks.

6. Ensemble simulation of many cases at different sites and subsequent collation of results centrally.

7. Simple meta-computing jobs.[4]

The next section discusses some features of Globus that are not provided by Unicore and therefore represent additional motivation for integration.

[4] Unicore provides abstractions for resource and coarse grain job synchronization (e.g. hold jobs until a given time.

2.2. Globus

Ian Foster and Carl Kesselman described the Globus project in it's earliest stages as:

> *The Globus project is attacking the meta-computing software problem from the bottom up, by developing basic mechanisms that can be used to implement a variety of higher-level services.* *[Fost97]*

This is clearly a contrast to the top down approach taken by the Unicore system. The resulting Toolkit provides the potential of an almost universal level of functionality, compared to the narrower focus of the Unicore environment.

The Globus Toolkit provides many important Grid components and services, including a security infrastructure (GSI), Metacomputing Directory Service (MDS), secure third-party file transfer, and a Resource Description Language (RSL). Globus Job-managers can be configured for all popular batch subsystems, including PBS, LSF and Condor. Through MPICH-G2, a communications infrastructure for meta-computing jobs is also available.

In particular, there are two components of the Globus Toolkit that provide real strength, that contrast the Unicore approach, and therefore form part of the motivation for the integration of Globus and Unicore. These are the CoG (Commodity Grids) Kits [Lasz01], which provide an application level API, and the way in which the MDS services support a dynamically changing grid.

The Globus CoG Kits provide an interface to the application developer that allows extensive use of the Grid infrastructure. These components can be used to simplify the construction of complex jobs in RSL, and can be used to launch jobs (via GRAM and implicitly via DUROC through full delegation) which work in a tightly coupled meta-computing environment. With Unicore there was a deliberate intension not to support tightly coupled meta-computing or application level development explicitly. The target users had existing applications that did not use Metacomputing environments. These applications were frequently provided by third parties.

The Metacomputing Directory Service (MDS) provided by Globus is based on the assumption that the computational resources on a Grid are dynamic in nature. There is a significant level of intellectual sophistication required in the design of such information services. Until recently, the Unicore approach has, except for fine grain information such as machine load, assumed a static grid.

The overall combination of Globus ubiquity and advanced services with the work-flow support of Unicore provides a significant motivation for a combined solution. However, when attempting such a merger, careful analysis of the interfaces, where the two approaches overlap or provide redundant or competing solutions and where they differ, is necessary to the completion of such a project. The following section outlines how these interfaces interact to create an architecture for such a combined solution.

3. A Work-flow Portal for Globus

Both Unicore and Globus provide the conceptual cycle of resource discovery, job creation with a resource request, job adaptation for site specific details, job execution, and job monitoring and control. The names and mechanisms differ significantly, but abstractly these ideas remain constant.

In the following sections an overall architecture for this hybridisation is presented, along with an analysis of the significant interfaces that must be adapted or extended. In particular, security models, resource mapping, job mapping, and file transfer are considered.

3.1. Architecture

Figure 1 outlines the basic architecture of integration of Globus facilities into Unicore. The Unicore Client, Gateway, and Network Job Supervisor (NJS) remain largely unchanged. A new plug-in to the client[5] is needed to perform the grid-proxy-init function and create the temporary proxy certificate, which can be included in the AJO as a Site Specific Security Object (already part of the standard Unicore protocol). With this exception, the AJO is the same as would be sent to any Unicore Usite.

Because Globus already presents a unified view of it's resources, it is logical to model it as a single Usite, although this is not mandatory. Thus a Globus Usite represents a Virtual Organization [Fort02] and obtains its view of Globus resources through a GIIS[6]. Within the Globus Usite, each major Globus resource (usually a single host) is modelled as a Vsite[7]. At the entry to the Globus Usite, the unchanged Unicore Gateway will authenticate the connection using a Globus or Unicore issued certificate. The AJO is then dispatched to the NJS depending on which Vsite was selected by the user.[8] See later discussion on security.

[5] The flexible plug-in interface, which facilitates the rapid development of application specific GUIs, is a significant Unicore strength.
[6] In the full paper, we will provide a terminology correspondence table.
[7] A Vsite is a collection of (one or more) systems sharing a file space and user administration domain.
[8] There are plans to incorporate dynamic resource brokering into the Unicore system.

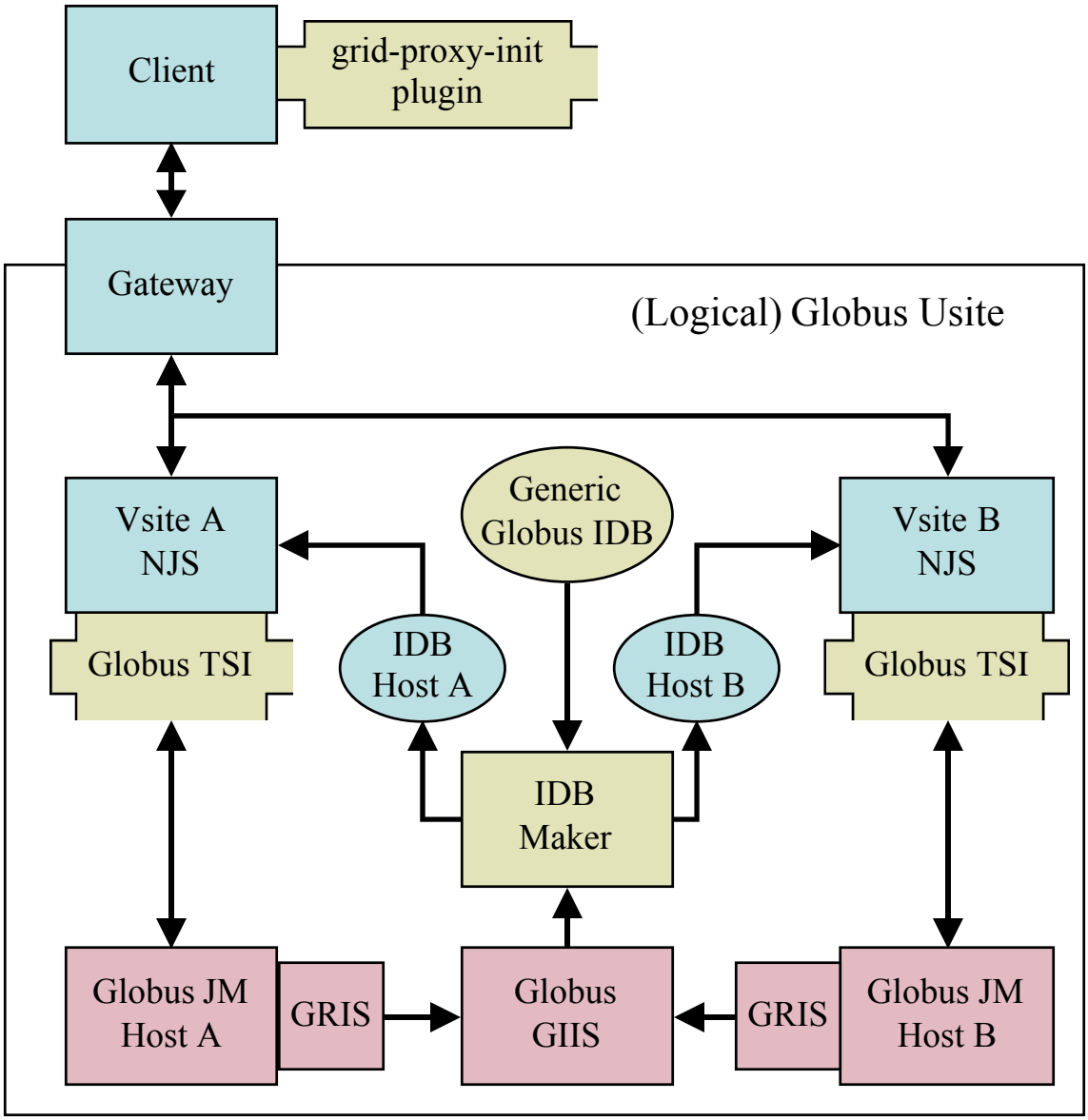

Figure 1: Overall architecture for Globus and Unicore interoperation.

The NJS also remains largely unchanged. Within the NJS, information from the host IDB is used to incarnate the job into a script. Construction of the script can be achieved with the help of the appropriate packages within the Java CoG Kit. The incarnated job is passed to a largely new component, the Globus Target System Interface (TSI). In existing Unicore sites the TSI is adapted to the peculiarities of batch subsystems on the various computational platforms. Traditionally, the TSI has been written in Perl, however, a Java based implementation is also being considered for this project. In particular, this would allow the TSI to utilise the CoG Kit.

The creation of the host IDBs is a major part of this strategy. Based on a Generic Globus IDB, and information obtained from the GIIS (or perhaps directly from the host's GRIS), a Globus Vsite specific IDB will be produced. These IDBs are created automatically by

the IDB Maker. The generic Globus IDB lists all those fields whose values must be provided by the GIIS (using GRIP via Java CoG[9]) in order to allow the successful incarnation of Unicore jobs by the NJS. Where this information cannot be obtained automatically, it may have to be provided manually.[10] Note that because Unicore's abstraction based approach limits what can be incarnated to those concepts for which abstractions have been created, the scope of this task is not as boundless as it appears at first. This interoperability problem will be investigated in future activities with the goal to achieve a unified standard that is promoted through the Global Grid Forum.

3.2. Security

Both Unicore and Globus base their user authentication model on X.509 certificates. Therefore, it is possible to use a Globus user certificate to run Unicore jobs as long as the servers are configured to accept the Certification Authority which issued that certificate. The reverse is also be possible, i.e. use a Unicore certificate to perform a 'grid-proxy-init'. This two-way certificate conversion has already been achieved. In addition interoperability has been achieved between certificates signed by the UK eScience Grid [Ukes02] which uses Globus certificates and the EuroGrid CA [Euro02] which issues Unicore certificates. In particular a Globus proxy certificate can be used to run a UNICORE job by a slight code modification to the NJS to ignore the proxy field and to validate via the original CA chain.

However, significant differences in the overall security models exit. In the Globus model, a temporary certificate is created by the 'grid-proxy-init' function and used as a proxy for a series of delegated actions within the Globus environment. This proxy is protected by standard Unix mechanisms at the individual sites, which is only a problem if these are compromised at the site. However, for one site to accept a proxy, which has come to it via another site, it must trust that site as well as the end user, i.e. "transitive-trust" is required. The Unicore model only requires the site to trust the end user but not any Usites *en route*.

These are not considered as fundamental interoperability barriers, since the security model relates to Grid policy operation, thus providing the fundamental interoperability of certificates exists as described above one can regard certificate-based authentication as a member of the intersection set of Globus and UNICORE services.

3.3. Resource and Job Mapping

The Unicore resource package is used for both advertising what resources are available at the Vsite and to describe a user's resource request. Within the Unicore resource object model, resources fall into three main subclasses:

---

[9] In [Fost02], changes in the protocols of Globus may be under consideration.

[10] Unicore makes extensive use of application meta-data which is not always provided by site's MDS services. The Grid Interoperability Project will make use of any emerging standards in this area, e.g. RIB [RIB02].

- **Capability Resources**: These represent services that are provided by the Vsite. Examples include application software, parallel libraries, and job priority levels.

- **Capacity Resources**: As their name implies, these resources all have a concept of capacity associated with them, e.g. memory, disk space, nodes, processors, etc.

- **Information Resources**: These resources describe other aspects of a Vsite that are not essential to the normal function of Unicore, e.g. contact address for site administration, system architecture, etc.

Although possessing a different structure, the Globus resource model, as described by the MDS, can be mapped to these three categories. It is not necessary that this mapping be isomorphic, as for this project the information flow is unidirectional. Globus resources are converted to Unicore resources, used to define a user's resource request, and then incarnated to run on a Globus Host. There is no requirement to describe all Unicore resources in MDS or to use all available MDS information in creating and incarnating a Unicore job. Since we are considering interoperability at the level of services the mapping does not have to be isomorphic. In fact one could envisage extra translation services on either side of the Grid exchange to cope with differing resource descriptions. Such translation services would be a useful pointer to a Grid Services ontology. Our feeling is that such an approach is more realistic than trying to create a single Grid Resource Description Language to which all Grid systems must conform. This is in line with the Grid Services approach, we describe resource description translation as a service with defined interfaces leaving open the possibility of multiple implementations.

There is detail in the MDS resource description (e.g. virtual memory size, current system load[11]) that is not used by Unicore and can be converted to information resources as a courtesy to users. At incarnation time, there is a system of default values defined for all Unicore resources that may be used in the absence of specific requests from the user. These defaults allow the incarnation of Unicore jobs, even where there are resources, required by Unicore, that are not published in the MDS, e.g. execution priority.

The combination of resource information from the GIIS, the defaults included in the host IDBs, and static mappings from Unicore abstractions (run script, cancel job) to corresponding the Globus functions, should provide all the infrastructure to complete the Grid Interoperability Protocol intended in this project. We also draw attention here to the importance of default values in interoperable Grid systems, they allow the interoperability transfer to work correctly even though incomplete resource descriptions (from the viewpoint of the system which is to execute in RP space) come across the "frontier" from the system which originates from RR space.

[11] As part of EuroGrid [Euro02] a resource broker is being developed to work in the Unicore environment. This facility will be able to take advantage of dynamic information from MDS.

3.4. Resource Discovery

One reason why the UNICORE initiation from RR space to Globus execution in RP space works well is that UNICORE can create dynamically as a service the IDBs by using the Globus MDS. Currently it would be impossible to do this the other way round since UNICORE does not have the hierarchical resource discovery mechanisms developed by Globus. This leads to the question as to whether resource discovery services should be in the minimal set. We consider that they should and in this sense the interoperability study has been of benefit to the UNICORE developers since it highlights an urgent question as to the scalability of UNICORE. The reason for this difference is probably that UNICORE set out to solve the problem of seamless secure access to heterogeneous computing facilities over a set of resources (the German Supercomputing Centres) which were relatively small in number (though very powerful) and which were known to the intended group of users. Globus has set out to solve the Grid problem in a more complete sense but in so doing has not focussed as much on the seamless and intuitive nature of Grid Access.

However this lack is not fundamental to the UNICORE model since the AJO concept allows resource discovery and indeed resource brokering to be considered as just another workflow request. In the EuroGrid project [Euro02] this idea has been used to integrate resource brokering without any change to the UNICORE model. We simply introduce the concept of a brokering Vsite and extend the AJO to include a Resource Check object. This allows the resources requested from the RR space to be checked against the resources in RP space known to the Vsite. The lack of proxy certificates in UNICORE makes a fully recursive resource search more difficult than it is in Globus since Vsites cannot take over the responsibility for handing on the request "What other brokering Vsites do you know about?" but there are ways round this. One is for Vsites to build tables of Vsites that wish to participate in the UNICORE Grid. An initial request from RR space to this Vsite returns this list which then is used to initiate more resource request queries going further afield. Thus the addition of the Resource Check object and the provision of brokering components on both the Client and NJS sides of UNICORE will give it Resource Discovery Facilities which would allow a job initiated from Globus to find resources on the UNICORE Grid. In this way Resource Brokering becomes integrated into UNICORE as an essential service within the model, in Globus such brokering facilities do not currently exist.

This brings us to an interesting aspect of the interoperability approach to minimal Grid services, namely it can be applied in an interative fashion. Had UNICORE stayed within its original "universe" this problem of resource discovery would not have been so clearly highlighted. Now this is being addressed via the inclusion of the resource brokering capability that allows the EuroGrid Resource Broker to check in advance that an AJO (from RR space) can run on all the multiple Vsites for which it is intended. This then becomes an interesting challenge to see if this should be regarded as an essential or highly desirable or merely optional Grid service in Globus. The fundamental question here is: should a well composed Grid request from RR space be able to check that it is potentially able to run on all intended resources before it is initiated? Note that this is

different from the issue of whether it actually runs because machines and network connections may fail during the consignment and execution phases.

We give details of the EuroGrid Resource Broker in Appendix A, in the Grid Interoperability Project it is intended to create an interoperable Resource Broker which can peform similar functionality on an RP space controlled by Globus mechanisms. The current broker status is that the code to implement the basic functionality described here has been released to EuroGrid partners. Testing and building of the EuroGrid implementation of UNICORE will take place in September 2002 with the intention of a stable release by the end of 2002.

4. Pointers to a Minimal Grid set

We now examine some indications of what a minimal set of Grid services might look like. Very simple approximations of the architecuture shown in Figure 1 have been realised to the extent that a simple Unicore job can be run on a Globus Grid using a Unicore certificate. We have not yet attempted multiple site jobs in this way and we have not attempted so far to implement a dynamic IDB. We believe that useful proof of concept studies can use a hand-crafted IDB. We wish to examine further the questions raised by OGSA before deciding how much of this architecture to implement or whether to adapt directly to Grid Services. IDB_maker would then become a service which might be generalised to a Grid Resource Request Translation service.

1) Authentication services, the concept of authentication of a request whether from a user or onwardly forwarded from another Grid service is vital. Given this concept we have found that two different security models using Digital Certificates based on X.509 can be made to interoperate. If one of the Grid systems did not have this concept then the other system could use a default option to give authentication automatically to the incoming request but this appears to totally break the abstraction of authentication so we would exclude it.

2) Ability to compose an object in RR (Resource Request) space. These objects may need to be translated from one system to another. For example a UNICORE AJO needs to be translated into a Globus script using RSL. Since this translation can be provided as a service itself it can be included as a subsidary service to the need to have the RR Composition service.

3) Ability to instantiate or “incarnate” (in UNICORE terminology) an RR object as an object in RP space. This can be done as a flow through both RR and RP space since resource requests received by site A may be partially translated in RP requests and partially forwarded on as a RR request to other sites. In this way of looking at things RR space can be more abstract than RP space and multiple mappings can occur. This seems to conform to current metacomputing usage of Grids but, as we have pointed out, distributed collaborative working may cause these spaces to be treated equally in abstract terms.

4) Ability to discover resources. The interoperability study shows that UNICORE currently lacks this service, we have described how UNICOREs AJO framework can incoporate this without any breaking of the UNICORE abstractions and as a bonus provide resource brokerage as part of the Grid services of UNICORE. This is due to the object-oriented nature of UNICORE which is realised in Java.

5) Notification of commencement and successful or unsuccessful completion of the RR request from RP space and return of all necessary information and output. We have not described this in the interoperability section but it is implicit in the architecture we have described since both UNICORE and Globus do this.

We have not included monitoring of the progress of the RR request in the RP space although both UNICORE and Globus can do this. Firstly we are not sure if this is an essential service, although it is certainly a highly desirable one, and would be included in a set of services for a usable Grid. Partly also we avoid this question because it brings up the question of how this monitoring is to be implemented. Is it polling from the originators of the object in RR space or notification from the providers in RP space or do we allow both of these on an equal footing. We think this question would be illuminated by an interoperability study involving Grids for dynamic collaborative working such as Access Grid. Here monitoring is persistent and mission-critical. We therefore propose to postpone the monitoring question to a subsequent such study. It is clear from our set-theoretic intersection approach to minimal Grid services that a single interoperability study can only provide pointers to the minimal set question. It is essentially the ensemble of such studies that is the true point of focus in our approach.

5. Conclusion

We propose to study the question of minimal Grid services by examining the interoperability of different systems describing themselves as Grids. We have given both theoretical and practical reasons for adopting this approach. A comparison of UNICORE and Globus proves to be useful since they are two systems which evolved independently to solve similar tasks, namely those of a Grid for large computations across distributed sites with autonomous security and access policies. This latter requirement focuses accessibility on the Grid onto the implementation of reliable authentication. We suggest an authentication service as a core Grid service.

By examining the way that Globus and Unicore work in practice we suggest that the concepts of a Resource Request (RR) space which provides for compostion of requests for Grid resources, allied with the mutually dependent concept of a Resource Provider (RP) space which allows us to describe how the requests will be implemented, are a natural basis for our analysis. This gives rise to two more essential Grid services, to create objects in RR and RP space. We draw attention to the possiblility of close

intertwining and interaction of these spaces, especially in Grids for distributed collaborative working. However in a simple "job submission" model we can localise the interaction to discrete and rare (in terms of overall time from job creation to termination) events. In large scale computations most time is in RP space. We draw attention to the need to translate between different ways of describing objects in RR and RP space. This could be addressed via a single mandatory Grid Resource Description Language, but we think the creation of a Grid Resource Ontology and Grid translation services are a better way to get Grids to interoperate.

Globus has Resource Discovery mechanisms which UNICORE does not. Logically our approach should lead to Resource Discovery not being an essential service. We choose instead to regard UNICORE as lacking in this essential service (for historical reasons) and describe how this is being addressed in the EuroGrid project. Since this extension, the provision of a Resource Check Object, does not break the UNICORE abstract model (indeed is completely harmonious with it) we do not consider that we have fundamentally violated the methodology of our study. If we had to bolt something onto UNICORE outside the model just in order to provide this functionality we would either have to conclude that it was not an essential Grid service or that UNICORE could not be described as a full Grid system. Considerations of scalability demand that the latter course be taken. If a user has to know the resources of all providers before job submission and the middleware cannot help her discover them, then the middleware does not possess sufficient abstraction to unify the resources in a Grid.

Finally we need a Grid service that handles the interaction between RP and RR space where the minimal functionality includes notice of commencement of the transfer from RR space to RP space plus notice of termination (with indication of status) from RP to RR including delivery of any objects (e.g. files) specified in the RR request. We propose to leave monitoring from RR as the work is implemented in RP as a desirable rather than essential service. There may be some Grid environments where such monitoring is essential, e.g. Access Grid, and others where it might be problematic (e.g task-farming).

Finally we suggest that other interoperability studies would be very useful for answering the question of what is the minimal set of Grid services. We suggest that a comparison between a job-oriented Grid (e.g. Globus or UNICORE) with an interaction-oriented (e.g. Access Grid, CAVE Grid) would be very informative.

6. Glossary

RR space – space of all objects that are requests for computational or information resource

RP space – space of resources that can instantiate or incarnate requests from RR space, e.g. processors, disk space, software, network bandwidth, …

AJO – Abstract Job Object, an abstract object in RR space provided in UNICORE by a Java class

VO – Virtual Organization, collection of resources grouped together to provide computational and information services to a community, this includes dynamical construction and joining of VOs

MiniGrid – Similar concept to VO but implicitly containing the idea that Grids have complex structure that is built by federating simpler entitities, the minimum entity being called a MiniGrid.

7. Bibliography

[Agdoc1] Access Grid Project – see collection of papers at www.accessgrid.org

[Broo00] J.M. Brooke, M. Foster, S.Pickles, K. Taylor, "MiniGrids: effective testbeds for Grid applications", in Proceedings of IEEE Workshop Grid2000, R. Buyya, M. Baker (Eds.), Springer LNCS 1971, 158-169, 2000

[Czaj01] Czajkowski, S. Frtzgerald, I. Foster, & C. Kesselman, "Grid Information Services for Resource Sharing", in HPDC-10, IEEE Press, 2001.

[Erwi01] Erwin, D. & D. Snelling, "Unicore: A Grid Computing Environment", LNCS 2150, Euro-Par 2001, Springer: 2001.

[Euro02] EuroGrid project, IST—1999-20247, www.eurogrid.org.

[Fost97] Foster I. & C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit", Intl J. Supercomputer Applications, 11(2), 1997.

[Fost98] Foster I. & C. Kesselman, ed. The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufman Publishers, 1998.

[Fost02] Foster, I, C. Kesselman, J. Nick, & S. Tuecke, "The Physiology of the Grid", Draft 2/3/2002, www.globus.org/research/papers/ogsa.pdf.

[GGF02] Global Grid Forum, www.gridforum.org.

[Glob02] Globus project, www.globus.org.

[Bive02] Bivens, H. Workflow drafts: www.sass3186.sandia.gov/~hpbiven

[Lasz01] Laszewski, G., I. Foster, J. Gawor, & P. Lane, "A Java Commodity Grid Kit" in CCPE, Volume 13, Issue 8-9, 2001.

[RIB02] Repository in a Box project, www.nhse.org/RIB/.

[Snel02] Snelling, D., Steering a Lattice Boltzmann Code via Web Services, preprints available from the author, snelling@fecit.fle.fujitsu.com

[Tent02] TENT project, www.sistec.dlr.de/tent/.

[Ukes] UK Grid Support Centres www.grid-support.ac.uk

[Unic02] Unicore information is available on the following web sites:
www.unicore.org - Unicore protocols and source code;
www.unicore.de - The home page for the Unicore Plus project;
www.fz-juelich.de/unicore-test - A complete Unicore environment where users can download a client, obtain a certificate, and trial the Unicore environment.

[w3c02] XML activity log www.w3.org/XML/Activity.

Appendix A: Resource Discovery and Brokering for Unicore

The Unicore software is a well established Grid solution which allows users to design complex componentised High-Performance Computing jobs, using a GUI interface to capture the workflow of the job from the user; the user can then run this job at any of their accounts on multiple heterogeneous Unicore-enabled resources. The seamless interface that Unicore provides means that retargeting a component of a job from a Cray T3E to an SGI Origin 3000 becomes a one-click operation.

Until now, Unicore users have had to manually target each component of their job at a specific Unicore-enabled resource, or Vsite. This solution, while acceptable for controlling access to a small number of resources, does not scale acceptably to tens or hundreds of resources. This paper describes a proposed solution to this problem: an advanced resource broker, capable of both resource discovery and resource brokering over Unicore-enabled resources.

Work on the broker has reached the coding stages. The primary source for this document is a technical specification, written and reviewed as part of the EUROGRID Project. Although the broker is being developed under the EUROGRID Project, it will be made publicly available as part of future releases of the Unicore software.

Overview of Unicore Architecture

Each Unicore-enabled compute resource is called a Vsite. Several Vsites at an organisation are grouped into a Usite. Each Usite runs a single Unicore Gateway process, which manages all incoming connections to that Usite's Vsites. Each Vsite runs a Unicore NJS (Network Job Supervisor) and several lightweight TSI (Target System Interface) daemon processes.

The user uses Unicore using a client which talks UPL (Unicore Protocol Layer) to the Vsites' NJS processes (via the Usite Gateway processes). A second protocol, called FNTP (FECIT NJS to TSI Protocol), is used by the NJS to control the TSI daemons.

This architecture has two particular advantages:

1. The load on the Unicore-enabled compute resources is negligible, as only the lightweight TSI daemons, written in Perl, run on the compute resource. The Gateway and NJS processes are Java programs but run on other, cheaper resources, e.g. Linux workstations.

2. This configuration routes all incoming connections to the Unicore Vsites via a single port on a single machine, which has been advantageous when using Unicore at sites with firewalls.

The user composes their job into an Abstract Job Object. This is essentially a hierarchy of components, such as scripts, compilation tasks, file transfer tasks, or application-specific components. The hierarchy is deepened by adding Job Group components to the AJO. Job Groups have their own components, and may include further Job Groups. Each Job Group (including the top-level Job Group) also has a workflow dependency graph to determine the execution order of its components. Although this used to be a Directed Acyclic Graph, this has recently been extended to include conditionals and loops, rather than just simple dependencies. Each component of the job can be assigned a set of resource requirements. These requirements can relate to number of CPUs, memory or disk usage, as well as software requirements, such as the presence of an ANSI Fortran 77 Compiler, or the CPMD Molecular Dynamics code.

Prior to submission, each Job Group is targeted to a specific Vsite; the Vsites that a job is targeted at do not have to belong to the same Usite. (File Transfer job components are used to transfer files from Vsite to Vsite.) The resource requirements are checked against the resource capabilities of the Vsites. Currently, in Unicore 3.5, the client performs this checking. However, with an internally used version of the software, the NJS processes can now perform this checking.

Each AJO job group is then signed by the user's private key, and dispatched to the Vsite that the top-level Job Group was targeted at. This **primary Vsite** is responsible for executing the components of the job that are to run locally, and dispatching the non-local job groups, while respecting the workflow dependency graph. This Vsite will also return the outputs of the job to the user.

Each Vsite executing parts of the job must have the user's public signed certificate. This is used to verify that the user signed the AJO part that they receive. Currently, the distinguished name from the signed certificate is also used by the NJS to determine a UNIX username under which to run the processes generated for the job.

Pallas Client

Currently, most Unicore users use the client from Pallas software. This client supports a Plugin interface which allow the user to write application-specific portals, e.g. for Chemistry codes such as Gaussian98 and GAMESS. The portals consist of a Java methods (delivered in a signed JAR file) which use JPanels to obtain inputs from the user, and construct a AJO component expressing the user's job requirements and resource requirements (e.g. must run at Vsite running Gaussian98). Currently, portals for Gaussian98 and CPMD have already been written, and are in use today.

Seamless Computing with Abstract Job Objects

One of the most interesting features of the Unicore software is the notion of seamless computing. The user need never specify the precise location of executable files for their jobs while constructing their AJO. Instead, they simply specify that they require a specific software resource, e.g. that a CPMD Molecular Dynamics job requires V3.4 of the CPMD software.

When the NJS prepares the job for execution, it will make specific the references to the software resource. This specialisation process is called **incarnation** and is controlled by a Vsite-specific Incarnation DataBase, or IDB, which contains this type of application metadata. An entry for the CPMD Molecular Dynamics code might look like this:

```
SOFTWARE_RESOURCE CPMD  V3.0h

SOFTWARE_RESOURCE CPMD  V3.4.1


TEXTINFORESOURCE [ The  CPMD Pseudopotential Library ]

  TAG     [PP_LIBRARY_PATH ]

  VALUE [ /usr/local/cpmd/lib/PP_LIBRARY ]


INVOCATION CPMD-V3.0h [

  PP_LIBRARY_PATH=/usr/local/cpmd/lib/PP_LIBRARY; \
```

```
    export PP_LIBRARY_PATH;

  /bin/mpprun        -n        $UC_PROCESSORS        \
    /usr/local/cpmd/bin/cpmd30h.x          $CPMD_FILE
    $PP_LIBRARY

]



INVOCATION CPMD-V3.4.1  [

 PP_LIBRARY_PATH=/usr/local/cpmd/lib/PP_LIBRARY; \

    export PP_LIBRARY_PATH;

 /bin/mpprun        -n        $UC_PROCESSORS        \
    /usr/local/cpmd/bin/cpmd3.4.1.x    $CPMD_FILE
    $PP_LIBRARY

]
```

Seamlessness then, is completely dependent upon there being standard ways of representing a software resource across all Unicore-enabled resources. Fortunately, most software resources are either standard resources defined in the default installation, requiring only that site-specific fields be altered, or are extensions that must be added for a plugin, when full instructions for adding the resource description are provided as part of the plugin documentation.

This abstraction of requirements, which separates requirements from platform-specific details enables seamlessness to function. This system is extensible, with Unicore administrators being able to define their own software resources. This enables the addition of site-specific software resources. By composing jobs in this way, job parts may be retargeted by simply clicking on a different Vsite when editing the relevant task group.

Description of Interfaces

Currently, in Unicore V3.5, the user manually selects a Unicore-enabled resource, or **Vsite**, for running each part (i.e. **job group**) in their job. It is then the responsibility of the client software to verify that the resource requirements of the job group are met by the target Vsite. This approach, although viable for a accessing a small number of Vsites, does not scale to hundreds of Vsites for two reasons:

1. Discovering sites with the required resources becomes difficult;

2. Deciding which of the possible resources is best is complicated.

These two problems will be addressed by the EUROGRID Resource Broker. The user will use the resource broker in the following way:

1. User specifies the Resource Broker as the destination Vsite for each part of the job which they want to be brokered (typically all parts).

2. During the Resource Check, prior to submission, the Broker is contacted, and discovers suitable execution plans for the job.

3. These are presented to the user, who selects a plan (if more than one is returned).

4. The Vsite information is copied from the execution plan into the job.

5. Only the brokering request goes through the Broker – the user submits the job directly to the Vsite(s), as before. All job-related file transfers and/or data streaming happen directly between the client and the Vsite executing the job.

This structure has the advantage of allowing the underlying job submission procedure to be radically altered, without a significant change to the user interface. In addition, virtually no modification to the UPL protocol was required; a broker is simply a Vsite that supports the Broker software resource.

Because of the Unicore architecture, whereby the UPL protocol is used between the Client and NJS processes, and between one NJS process and another (when submitting sub-AJOs), it is possible to place the resource broker either at a Unicore Usite or in a client Plugin, while using the same code.

The other possible job submission mechanism would have involved the user submitting jobs via the Resource Broker. This simplifies certain issues, including the notification of the broker of the outcome of the job, and charging issues where the broker is to receive a commission on the job. However, with the current Unicore architecture, this would cause file transfers and file streaming to be directed via the broker(s). This is unacceptable, both for performance reasons and because it would introduce a new point-of-failure into the system.

The individual steps of the brokering process will be described more fully in the final paper.

Implementing Brokering Functionality

As mentioned above, Resource Checking will be performed by the NJS Processes. To do this, the NJS will respond to a ResourceCheck method, sent by the client, along with a set of resource requirements. An additional method, called QoSCheck, will be provided as a more advanced form of resource check, taking the same inputs, but returning a QoS ticket, rather than a simple yes or no answer. The QoS ticket will contain execution plans. Initially this will contain the name of a Vsite, and an estimation of both

turnaround time and cost, based on the priority of the job, as well as its software, computational and disk requirements. A client will be able to tell whether or not a Vsite implements the QoSCheck method by looking at the software resources provided by that Vsite.

A resource broker then, is a Vsite which can return QoS tickets containing execution plans for executing the job or job-part at other Vsites (also, a broker can be part of a Vsite that can execute jobs, or it can be a stand-alone facility). "Normal" Vsites will only return QoS tickets for their own site. Some non-brokering Vsites will not provide estimates of run-times, and will only perform checking of static resources against the resources that Vsite provides, i.e. they will only implement the ResourceCheck method. These "dumb" Vsites impede the quality of estimates that can be returned to a user, and it is hoped that this type of Vsite will be phased out as underlying Batch Submission Systems gain the ability to estimate turnaround times for individual jobs. If this is not forthcoming, it may be necessary to develop software which can estimate turnaround time based on the current queue state, and past behaviour.

As well as brokering for the job part submitted to it, the broker will also be responsible for obtaining estimates for job groups within that job part, and for combining this information into the execution plans. With the current version of the Unicore NJS, a job part is only queued after the parts which it is dependent on have been executed. This model will be used by the Broker to combine the estimated run-times into different execution plans.