

## Abstract

The grid paradigm is summarised: the Virtual Observatory is shown to be a grid, whether or not it uses software products specifically built for grids.

Available middleware is reviewed for the following areas: computational services; contextual web services; data grids; access control. In all cases, the grid middleware is found to be a partial solution at best to the VObs problems. In many cases, the partial grid solution overlaps with and is incompatible with emerging standards from the IT industry.

The best use of the grid middleware seems to be at the edges of the VObs' tree of services. Here, it can possibly allow access to special resources for exceptional use cases. Most grid middleware is not well suited for the mission-critical parts of the VObs.

However, some aspects of grid software fill a need and could be made pervasive in the VObs. The GridFTP protocol and the grid security infrastructure are the strongest candidates.

## What makes a “Grid”

There exists special middleware written for making ‘grids’. It is different from the middleware that Virtual Observatory (VObs) projects have used to date. Is this middleware useful to the VObs? What does it achieve?

A grid is a collection of on-line services with a few special features not found in the world-wide web or in typical web-service installations.

- There is a “market” of commodities: services with equivalent semantics that can be used interchangeably. Use of these services may be free, or it may be controlled by barter or the exchange of money. The services are listed in registries.
- The services carry out long-running operations: loosely, “batch jobs”. These jobs can run for longer than the typical up-time of a machine in the grid, so cannot be controlled via a chain of web services that remain connected for the duration of the job. Instead, the grid infrastructure allows the clients and agents in the chain of command to start the job; to detach from the services doing the actual work; and to reattach later to “steer” the job or to retrieve its results. This asynchronous, detachable mode of operation dictates the architecture of the grid. It shapes the operations of the services in the grid and dictates that the grid itself stores the results of its job steps, rather than sending them to the desktop for storage.
- Services doing practical work in a job are supervised and coordinated into workflows by other services further up the chain. Results from one service are fed as inputs to others in the workflow. To support this, a peer-to-peer system for exchanging data is provided as a separate feature of the architecture.
- Since data in the grid are sometimes private, and the resources in the grid are valuable and scarce, access to data and services is controlled. To preserve the detached mode of usage in a secured grid, there is a single sign-on system in which the users can delegate privileges to software agents. The extent of the

single-sign-on system defines a Virtual Organization (VOrg)<sup>1</sup> that typically includes more than one conventional organization.

The virtual observatory (VObs)<sup>1</sup> aspires to all these features. It is clearly a grid, even if it is not built from branded “grid technology”.

Already, within VObs projects around the world, these features are being provided. The principal question is whether the VObs grid can be built better by using products specifically designed for grids.

### ***Different grids for different commodities***

There are different commodities that are shared on grids: algorithms, CPU cycles, data-storage space and mirrored copies of particular data-objects.

A grid of services, called in client-server fashion, supports the sharing of algorithms. This kind of grid has three benefits: clients can run the algorithms without the cost of installing the software locally; the algorithms can be run at the site of the data they process, which conserves network bandwidth; and the algorithms may run on faster computers than the user’s client computer. This network of services becomes a grid when algorithms become commodities, such that users and their agents can choose the best site for the job from a registry of services that do the same work and have the same interface.

The services in the grid may be “job manager” services. These submit batch jobs to processor queues. In this case, the shared algorithm is the command shell of the underlying operating-system. By sharing this, the services make CPU cycles a commodity; since the user, not the service, defines the computation. A network of job-manager services becomes a grid when the services have common interfaces and are catalogued in a registry.

Data-storage is shared by allowing either user agents or services to read or write files to any grid location where they have access rights. If only the user agents have this access, then data can only flow between services via the user’s desktop, which may limit the speed of execution of a workflow. When the services have the same access, peer-to-peer sharing of data is the norm, and this allows the system to exploit fast network paths between service sites. As a bonus, coupling between entities in the grid is reduced. A data-sharing network becomes a data grid when the storage sites are catalogued in a registry, when all the parties in the grid can access the storage through common interface and when the system has a common way of expressing access rights for users. Currently, most data-grids just store files. Later grids may give the same access to database tables.

Data-mirroring systems maintain copies of files to increase availability and reduce the need to fetch remote files over limited network-bandwidth. Mirroring is a well-established technique, but, in typical installations, only a few applications know the location of the mirrored copies and the meaning of the file. If the system can express and record the meaning of the files and the relationships between the copies, such that

---

<sup>1</sup> It is unfortunate that the grid community takes “VO” to mean “Virtual Organization” while the astronomical community understands it to mean “Virtual Observatory”. The confusion is increased by the fact that a Virtual Observatory is a specific kind of Virtual Organization. In this document I will use the abbreviations VOrg and VObs for clarity.

all applications can use this information, then the files themselves become commodities and the system becomes a grid.

A grid is only worthwhile if the commodities it shares have greater value than the cost of building and operating the grid. The first grids were networks of job-managers services sharing CPU cycles, since CPU power was seen as the most valuable commodity and the easiest to share. Grids sharing specific algorithms in custom services received less attention, since algorithms were seen a less valuable than CPU cycles. Proper data-storage and data-mirroring grids are extremely rare; access to data is recognized as a valuable commodity, but the data grids are harder to build.

Practical grids combine the techniques and share more than one commodity. Most grids sharing CPU cycles also share files, although few provide a full data grid. Data grid implementations generally provide both mirroring and storage. Therefore, most grid toolkits address parts of all the problems.

Computing for observational astronomy is limited more by data volume, data complexity and the difficulty of setting up software applications than by lack of CPU cycles. Most use cases can be enabled by installing affordable, extra computers at the sites where the relevant data are stored. Therefore, the “traditional” CPU-sharing grids have limited value, and grids of services in the VObs are more likely to contain algorithmic services than generic job-manager services. Conversely, numerical modelling in theoretical astronomy is entirely limited by CPU power and benefits directly from CPU-sharing grids; but the VObs is not funded to support theory work.

### ***Intragrids, extragrids***

Given an existing network of services, there are two ways to add grid products.

In an “intragrid”, components and services forming a complete grid are hidden behind a façade that conforms to the VOrg’s existing conventions. Typically, a grid is attached to one web service in the VOrg as a private resource. Users in the VOrg get the power of the grid by addressing the façade service. Applications and middleware in the general VOrg are not aware of the presence of the intragrid.

In an “extragrid”, the grid components, with their conventions, standards and protocols, are visible to applications; the grid parts are listed in the VOrg’s registries. Applications and middleware in the VOrg have to adapt to the grid software in order to use the resources it controls. The VOrg becomes dependant on the grid technology.

A network that is an intragrid for one VOrg may be an extragrid for others. An example of this is the TeraGrid which is clearly an extragrid to its member institutions: the processing power is distributed between several legal organizations. The AtlasMaker service for the VObs uses the TeraGrid as an intragrid for its own purposes of combining images, but does not allow other VObs entities to start arbitrary computations on the TeraGrid.

Intragrids are clearly easier to introduce to the VObs than extragrids but they dilute the value of grids. To get the most benefit from the grid paradigm, the VObs should standardize on one set of grid middleware and that set should become an IVOA standard for the extragrid. In addition to this, and in the interim before IVOA makes it choices, intragrids can be used to try out the available products. These intragrids need not use common technology, and there is no need to make the middleware for the intragrids interoperable; each intragrid only needs to interoperate with the façade service that links it to the VObs.

Ultimately, technology for intragrids is chosen by service providers and technology for the extragrid in the VObs is chosen by IVOA. AVO need not and should not dictate choices to the service providers and can influence, but not dictate or ignore, the decisions of IVOA. Once IVOA has chosen the standards for its extragrid, AVO may only choose amongst compatible implantations of those services.

## **Software for computational grids**

### ***Web services***

In the beginning, grid middleware was defined in the traditional way for Unix services:

- servers running as daemon processes, with one daemon per service;
- network communications via TCPP/IP sockets;
- custom communications protocols layered on TCP/IP;
- one TCP port-number per protocol; hence one port per service.

These methods limit development by forcing application programmers to work at too low a level and by restricting the reuse of communications code. Very little of the middleware can be shared between different communications protocols.

These techniques are now being superseded by web services, defined as on-line services communicating by exchanging XML documents over established network-protocols such as HTTP. The use of XML allows much richer communication, which supports services with complex semantics. The use of standard protocols at a higher level than TCP/IP greatly eases the development of clients and services.

Web services use XML by definition, but can have many different ways of using XML. Effective middleware requires standards for the logistical packaging of messages and for the description of the messages that a given service accepts. The IT-industry standards for this are the Simple Object Access Protocol (SOAP) and the Web Services Description Language.

WSDL can define the syntax of the messages of any web service insofar as the level of XML elements with given types. It can also describe the pattern of messages – e.g. request/response, unacknowledged transmission from client to service, asynchronous notification from service to client – for a given operation. WSDL allows communication with the service to be defined in terms of XML schema. WSDL is precise enough that communications code for talking to a service can be derived from that service's WSDL. The definition of commodity classes for services – i.e. groups of services that can be used interchangeably – is made easier by WSDL.

WSDL is the glue that binds services together. It allows clients and services written in different languages and with different programming-toolkits to interoperate reliably. Services that lack WSDL only interoperate by chance.

WSDL defines the basic schema of messages used by a given service. SOAP defines a wrapper or “envelope” for that schema that is common to all web services: the service-specific payload within the envelope is called the “body” of the message. In addition to the body, SOAP messages may carry various forms of header, and these are used to express logistical aspects of the transmission of the message: e.g. security

credentials of the caller, routing, encryption for privacy, and the contextual relationship with other messages in the same operation.

Bodies of SOAP messages can be literal XML documents, controlled by schemata provided by the author of the service, or they can be calls to object methods in the service, encoded according to a schema that is part of the SOAP standard; the latter is called Remote Procedure Call (RPC) encoding. RPC-style messages were the original aim of SOAP (and the origin of its name), but are now considered an impediment to interoperation. Document/literal messages are favoured for modern services.

The canonical web service has therefore become one which:

Communicates using XML;

Uses a standard, medium-level protocol to move the bytes, normally HTTP;

Describes its messages and patterns of messages using WSDL.

The VObs *should* be building services of this kind for its extragrid. To do otherwise would make the services and clients harder to write and the interoperation harder to arrange. IVOA has already agreed to base its grid on web services.

We should note, however, that the web-service paradigm applies mainly to control messages in a client-server situation. I.e., web services are ideal for invoking resources in the computational grid and for moving metadata. They are less well suited to moving bulk data. For the movement of bulk data in the data grid, we should look to other kinds of protocol.

Web services need not be SOAP services; there are other ways to package messages and the two simplest are already in common use in astronomy.

HTTP-get: invoke a service via its URL and embed the parameters of the invocation in that URL; receive the results of the invocation as an XML document; the return document is entirely specified by the service with no wrapper layer that is common between services.

HTTP-post: invoke a service by sending an unwrapped XML document; receive another unwrapped XML-document in return.

Both can be described in WSDL; both are simpler to implement than SOAP services. Why then use SOAP?

SOAP is unnecessary for services with simple semantics that are used in isolation. When services are combined in context, then SOAP headers become useful. These contexts are defined by the middleware layers that combine the services; authors of services may not be aware of the contexts when the services are written. Therefore, VObs services should use SOAP in order that the contextual headers can be added later without disrupting the interfaces of the services. The next section discusses the addition of context to services

### ***Contextual web services and OGSi***

Simple web services are synchronous, stateless and return results over their control channel. I.e., each invocation of a service blocks until the operation is complete, returns the results of the operation to the caller and then forgets the context of the operation. Subsequent invocations take place in a separate context. If the client breaks connection before the operation is complete, then the results of the operation are lost.

Synchronous services are useful for quick operations because they are easy to implement. The grid, however, calls for long-running operations where the caller is not continuously connected. Grid services commonly need to be asynchronous. This means that grid services need to remember the state or context of an invocation such that the caller can later retrieve the results. They also need a way to notify the caller asynchronously of progress.

The Open Grid Services Infrastructure (OGSI) is GGF's standard method for allowing web services to remember the state of a transaction and hence to run asynchronously. Web service context (WS-CTX) is the IT industry's draft standard for the same function; it is an OASIS standard. The two standards are fundamentally incompatible.

OGSI-compliant services are SOAP services that distinguish contexts by assigning a new URL to each context; they require that the application software set up this URL explicitly using a factory service. OGSI complicates the semantics of all grid services but leave the syntax of the invocations simple.

WS-CTX-compliant services are SOAP services that distinguish contexts by notations in the headers of their messages. They complicate the syntax of messages in order to keep the semantics of the service simple.

OGSI defines a general notification-interface for the client over which any message can pass. It does not define a standard protocol by which a service can signal the completion of an asynchronous operation.

WS-CTX defines a specialized notification-interface for the client with a specific way of signalling "job complete". It does not support notification of arbitrary events.

Some higher-level protocols may be needed for some use cases. Notably, there may be a need for web-service operations to be made "transactional", such that either all operations in a workflow succeed or all services are "rolled back" to the state they had before the start of the workflow. The WS-Transaction (draft) standard of OASIS defines protocols for this; it is built upon WS-CTX. OGSI has no equivalent protocol.

Parastatadis et al. [ref?] have compared OGSI with the OASIS standards. They conclude that OGSI is only a partial solution to contextual services; that complicating the semantics of the services (OGSI) causes more problems than complicating the syntax of messages (WS-CTX); and that using OGSI deprives grid services of support from most industry standards and the tools that follow those standards.

It is unlikely that the IT industry will abandon WS-CTX and switch to OGSI. Some industrial players may support both standards, but they are likely to put more effort into supporting WS-CTX. OGSI, in its current form, may or may not survive. If it does survive, then it is likely to be restricted to the scientific community.

If the VObs wants to write contextualized web services, then these services should not use OGSI but should follow the industry standards. If the VObs wants to use grid services written in the grid community, then it will have to write OGSI clients-software.

## ***OGSI and OGSA***

The Open Grid Services Infrastructure (OGSI) was introduced in the previous section. It is a standard for expressing "state" or "context" in web services.

The Open Grid Services Architecture (OGSA) is a factoring of generally-useful functions into standard, commodity services.

Currently, OGSA is pure vapour. OGSA will be a GGF standard, but no OGSA services have yet been defined. The areas to be covered are [list here...]

OGSA is based on OGSi; OGSA components can only function properly in a grid with OGSi semantics.

### ***OGSi implementations***

OGSi is defined by its WSDL and XSD schemata; services can be coded directly to this standard without requiring special tools or libraries. However, OGSi is sufficiently complex that most authors of grid services work with a reusable “toolkit” that implements OGSi with a relatively simple set of APIs.

There are currently 5 [?] OGSi implementations.

GT3

University of Virginia

MS.NETGrid from EPCC (<http://www.epcc.ed.ac.uk/~ogsanet/>)

Fujitsu Unicore.

OGSi:Lite (<http://www.sve.man.ac.uk/Research/AtoZ/ILCT>)

## **Software for data grids**

### ***GridFTP***

The conventional FTP protocol [ref?] is sufficient for moving files between services when: the data are in the public domain and may appear on a public file-server with no access control; the transfer is driven by the receiving server (a ‘pull’ transfer). I.e., if anonymous downloads are good enough for a use case, then conventional FTP is good enough.

For uploads (‘push transfers), and for restricted downloads of non-public files, conventional FTP uses passwords. This is unacceptable in a grid: software agents start the transfers and a user may not be available to give the password.

The GridFTP protocol, which is a GGF standard, extends conventional FTP with GSI security such that the security tokens used in the computational grid can also be used in the data grid in place of passwords. This enables the computational grid to drive the data grid.

Grid FTP also has features – transfer “striped” over multiple connections and control of buffer sizes – that improve performance with respect to conventional FTP.

Transfers in the data grid must be robust against network failure. It is inefficient to restart a transfer of 10TB from scratch because of a network failure halfway through. A grid needs a reliable file-transfer service and that service needs to be able to restart transfers. GridFTP provides the ability to restart transfers from ‘markers’ in the data stream.

A GT3 server is provided in the Globus Toolkit. Client software for C is provided in the Globus Toolkit itself and for Java in the Globus Commodity Grid Kit for that language.

## **Reliable File Transfer Service**

The Globus Toolkit version 3 includes a web service that controls transfers between GridFTP servers. The service remembers transfer state (even across restarts of the service itself; it uses a RDBMS for persistence) and restarts failed transfers.

The Globus RFTS uses GridFTP to move the files.

No tests have been made on this implementation, but the basic idea is *necessary* for running the data grid.

## **Replica managers**

Several proposed.

One is available with GT3. This is based in part on EDG work.

GGF is proposing other RLS for standardization.

## **OGSA-DAI and LDAS**

OGSA-DAI, from EPCC [ref] is an interface from the grid to relational and XML databases. It is a set of OGS-compliant web-services of which the Grid Data Service (GDS) is the primary part. A GDS instance represents a database connection and the associated state: e.g. it retains enough state to support a cursor on the results of a query.

When OGSA-DAI is used with relational databases, which would be its main use in the VObs, it acts as a JDBC redirector with added security. A request from a grid user is authenticated using Globus message-level security (see below). OGSA-DAI maps the grid identity of the user to a local user of the database and forwards the query over JDBC to the separate DBMS. OGSA-DAI does not itself provide or require a specific make of RDBMS.

OGSA-DAI does not, by default, do any transformation of the query or the results. Therefore, it does not abstract away the implementation details of the DB. This makes it unsuited as the primary interface to relational archives in the VObs. Data publishers in the VObs could force OGSA-DAI to do the transformations necessary for a VObs-standard interface [ref: skynode], but it would be much easier to use a web service with a conventional JDBC connection to an archive database. However, two features of OGSA-DAI make it useful in the VObs data-grid for special cases: access control for writeable databases and GDS-to-GDS transfers.

OGSA-DAI supports normal Globus-style access control. This makes it relatively easy to grant grid users write access to selected databases. It should be possible for users to upload tabular data in a VOTable into a GDS and use on those data the full querying-power of the RDBMS. Clearly, this feature could be coded specially for the VObs, but OGSA-DAI has it in reusable form.

OGSA-DAI allows the results of a query in one GDS to be written directly to another GDS without first being exported to a file and explicitly moved between sites. This supports the operation of a distributed data warehouse, which may become essential for large-scale data-mining in the VObs. The results of initial queries on many, scattered archives can be collated in a data-mining centre where queries with table joins and whole-table scans can run more efficiently. The final results can be exported to yet another database for long-term storage. Again, these facilities could be coded specifically for the VObs, but OGSA-DAI already has them.

For the data-warehouse scenario, OGSA-DAI can be used with a separate, web-service façade that handles the astronomical details of the work, leaving OGSA-DAI to handle the application-independent parts. I.e., OGSA-DAI is being used as part of an intragrid. In this mode, the heavy data-mining can be delegated to borrowed resources outside the normal VObs, e.g. on a national science grid.

OGSA-DAI was the original implementation. ELDAS [sp?] is a reimplementaion of the same specification by edikt and may have better performance than the original; ELDAS is supposed to be fully interoperable. The DAIS working group of GGF [ref] is working on an expanded specification based on the OGSA-DAI concepts.

### ***Storage resource broker***

Storage Resource Broker (SRB) [ref] combines reliable file-transfer and file-mirroring to make a self-contained data-grid. It also allows files held remotely on the grid to be opened directly by programmes, rather than explicitly copied and opened locally.

In SRB, a network of agents – brokers – manage all storage in the data-grid using custom protocols. Programmes use files via a custom interface similar to the C standard-IO library, and all files appear local to programmes. Files are copied and cached by the broker network as needed to maintain the illusion of local files. The location of files is stored in a central database and the broker network refers to this database.

SRB has powerful administration tools for making mirror copies and for migrating files between storage locations. It is particularly strong in moving files to and from tertiary storage; SRB is a great aid to running robot tape-archives.

SRB is very powerful, but it significant drawbacks.

- It is a closed system. Source-code is provided, but no third-party products can work on the stored files. The details of how files are stored are a private part of SRB.
- The file-metadata database is central and cannot be mirrored. If that database is unavailable then the entire data-grid fails.
- The ability to open and access remote files only works for programmes compiled with SRB's special APIs. Other software would have to be rewritten to use this feature.
- The central catalogue of files is naturally owned and controlled by one site in a grid. It is possible to federate SRB installations for adjacent grids, but a site wanting to join an existing SRB grid must give up control of its grid-attached storage.
- SRB deals only in flat files. It does not work with relational data in the way described for OGSA-DAI.

These drawbacks suggest that SRB is not well suited to extragrids and is not appropriate for the extragrid in the VObs. However, it is ideal for individual VObs sites with distributed storage and may be useful for building intragrids at VObs sites.

## **Access control**

Most grid operations involve changing the state of one or more services and this makes the services more vulnerable to misuse than would be the case with stateless services. A grid needs to allow access to vulnerable services only for authorized users. With conventional web services, this access control is exceptional; for a grid is the norm.

Authorization checks are pointless unless the user's identity is known. Access-control systems need to provide authentication of identity on all accesses to services. This in turn requires a system for registering identities that spans the entire grid. The limits of a grid and a VOrg are defined by the range over which its users' identities are recognized and trusted.

Traditionally, authentication is done with passwords specific to one site and authorization is done piecemeal within each site using file permissions. This does not scale to a grid; it is generally considered too hard to set up the local authorizations and too limit to prompt the user (who may not be continually present during a workflow) for passwords at each access.

Grid middleware is converging on a solution based on these principles:

- digital signature (based on public-key cryptography) for proving identities;
- public-key infrastructures (PKIs) for assigning grid-wide user names;
- authorization servers for sharing knowledge of users' privilege around the grid.

These techniques combine to provide 'single-sign-on' operation of the grid where a user only supplies a password once on starting each session.

### ***Public key infrastructure for grids***

Grid middleware generally implements authentication using the Grid Security Infrastructure (GSI); this is derived from prototypes in early versions of the Globus Toolkit. GSI requires the authenticating party to have an X.509 certificate containing an X.500 'Distinguished Name' (DN), and the DN is that party's grid-wide user-name.

A Certificate Authority (CA) – a third party trusted by all users and service providers in the grid – signs the certificate digitally, thus preventing forgeries. Before signing, the CA makes some checks, typically using real-world identity documents such as passports, of the applicant's identity, such that certificates are not issued falsely. The procedures by which certificates are signed and issued, plus the format in which the certificates are written, forms the Public Key Infrastructure (PKI) for that grid.

The quality of the off-line identity checks in a PKI defines the security of the grid using the PKI. If the security is very lax, then some potential service-providers may be unwilling to donate resources to the grid or to any network that interoperates with the grid. In particular, job-submission services are dangerous to use with a lax PKI as they give command-line access on grid nodes to anybody who can fool the CA. Algorithmic services are less vulnerable in this respect.

The minimum requirement for a party to use a secured grid service is that the party can present a GSI-compatible certificate with the request. There are many possible

uses of the X.509 format (the X.509 specification is regarded in the IT industry as unreasonably vague). It is currently unclear whether certificates issued by commercial CAs, such as Verisign, are compatible with GSI. Grids use special extensions to X.509 that are defined as part of GSI and discussed below under authorization. Without these extensions, some features of grids are unavailable. Currently, only CAs directly concerned with grid work, such as the UK e-Science CA, are known to produce compatible fully-certificates.

Thus, if the VObs is to use GSI, as it must to use standard grid middleware, then we must find a set of CAs covering all legitimate users. We cannot expect one existing CA to sign for all users of all nations; the global, commercial CAs may not issue compatible certificates; the national science CAs produce good certificates but will not sign for foreign nationals; and global, science-oriented CAs (such as the Globus CA, now discontinued) have insufficient identity checks when issuing certificates. For any given set of CAs, the overall security of the combined PKI is equal to the security of the weakest PKI.

Suppose that the VObs starts with a set of selected, trusted CAs which are GSI-compatible and which have sufficiently-secure PKIs. These are likely to be the CAs run by science-funding bodies in the industrialized nations. Choosing the core set requires agreements between all the major partners in the VObs and must be restricted so as to be 'politically' acceptable to service providers. The core set of CAs will exclude *many bona fide* users around the world.

In the medium term, the core set of CAs must be expanded to cover all possible users of the VObs. There are several ways this might be achieved.

1. Arrange with the global, commercial CAs to provide GSI-compatible certificates. There would be an annual cost for every certificate issued.
2. Run new, local CAs on behalf of the VObs. Initially, all VObs projects might have separate CAs and these could be made interoperable if all service providers in all VO projects agreed to trust all the VObs CAs. If some service providers distrust some CAs, then the interoperation is partial; this might be acceptable in the case of a few, special services. Users with no 'local' VObs project would have to make arrangements in another region or country.
3. Run one new CA on behalf of IVOA. Again, all service providers would have to trust this CA. Running an international CA to reasonable security standards is hard; it is difficult for users to present physical identification when they live on a different continent! It may be possible to delegate the identity checking to regional or national CAs. However, this fragments the trust model: service providers will not then trust the IVOA CA since they do not trust some of its subsidiary CAs.
4. Provide gateways between national grids such that users trusted in one grid can be authenticated in a peer grid. This means that operations in the peer grid are done in the name of a software agent from the requesting grid. The end-user's DN, if it is required for the operation, has to be sent by some other means than the certificate used for authentication.
5. As an extension of point 4, dispense with end-user certificates altogether. End-users authenticate to a web portal using a password, and the portal passes the job on to the grid using the DN and certificate of a software agent. All the work, even in the 'home' grid and PKI is done under the agent's credentials.

Option 5 is the only one that works in the transitional period when most end-users have no certificates. Therefore, this is the method used in the AstroGrid infrastructure, which has to work in the early days of the VObs. Option 1 is the simplest, both technically and sociologically.

### ***Authenticating requests to services: transport and message-level security***

GSI enables an agent sending messages to authenticate cryptographically to the party receiving the messages. This feature can be used either to protect the transport channel carrying the message, or to protect individual messages.

Traditionally, web services and grid middleware (e.g. Globus Toolkit 2) has used transport-level security; the web services were using HTTPS and the grid services were using custom protocol stacks GGF have standardized a set of APIs for making transport-secure connections using GSI. However, this standard defines only the interface, not the implementation or the wire protocol. The details of the security implementation are closely tied to the protocol of the data exchange. There is little reusable code for implementing transport-level security.

HTTPS is not ideal for web services. It encrypts all the data of every message, which is an unnecessary load in most cases; the communicating parties want authentication and integrity of data, not privacy.

Modern web services are starting to use message-level security, in which individual messages are digitally signed or encrypted as needed. Where authentication is needed, messages are signed or encrypted and authentication is implicit. In SOAP messages, the security provisions are put in the SOAP header. This makes them independent of the protocol in the message body. Code for securing SOAP messages can be reused for all such messages in all applications. There is an OASIS standard for this: WS-Security (<http://www-106.ibm.com/developerworks/webservices/library/ws-secure/>). It defines the schema for including signature and encryption metadata in the SOAP header.

There is currently no GGF standard for using WS-Security with grid services; it is not part of the OGS 1.0 standard. However, Globus Toolkit 3, which is widely seen as the prototype of OGSA, uses message-level security in preference to transport-level security; it does the former according to WS-Security and deprecates the latter.

VObs web services are clearly better off using message-level security according to WS-security. Message-level security is suited for secure communication with ordinary web services and also allows interoperation with OGSA.

### ***Avoiding 'replay' attacks: WS-Secure conversation***

If a sender signs each SOAP message, using the message-level facilities mentioned above, then the service accepting those messages is only partially secured. An attacker with access to the network can capture an entire, signed message and replay it to the service, thus posing as the original sender under the sender's signature. This can be used as a denial-of-service attack if the effect of the message is not idempotent.

To defeat replay attacks, the service can require a higher degree of message-level security as specified in the (draft) WS-SecureConversation standard (<http://www-106.ibm.com/developerworks/library/ws-secon/>). This standard uses the context of messages to render a message invalid if replayed out of sequence.

Globus Toolkit version 3 implements WS-SecureConversation as the default mode for secured services. (It can also use the basic signature and encryption modes if so configured.). It seems likely that this will become the eventual standard for GGF, since service providers will not be willing to lay their services open to replay attacks. However, WS-SecureConversation is not finalized yet, so it is unlikely that the current GT3.0 implementation will match the eventual standard, or that it will fully interoperate with implementations in other toolkits.

## **Authorization**

Authorization can be very simple or quite complex. The complexity of astronomical data and IPR suggests that the VObs needs the more-complete kind of authorization.

A very simple service may have implicit authorization: any user who authenticates has equal rights in the service. This presumes that all users are equal (and so does not allow different treatment by the VObs of, say, high-school students and professional astronomers) and equally trusted (requiring that all users of the VObs be vouched for by CAs of the same quality; see the discussion above). This authorization scheme is insufficient for the VObs.

Adding one step of sophistication, a service may map each grid user's DN to a local account. The privileges accorded to the DN are those accorded to the local account. This is the default scheme for the Globus Toolkit, which uses a local file, owned and maintained by the service provider to define the mappings between DN and Unix accounts. A comparable scheme is used in OGSA-DAI, which maps DNs to database users via a local file. This scheme works for special services with a few, privileged users; this is why it has survived in prototype grids until now. The scheme does not scale well to grids with many users, since every service provider must make local changes to the mapping file for each new user. The VObs may use this scheme for a few special services, but it will not do for general operations.

If nodes in a grid are very uniform, the authorization map-file may be mirrored from a central server. GridPP and EDG have used this technique with Globus Toolkit 2. This may be a useful stop-gap for the VObs projects, but the VObs as a whole is too heterogeneous and too decentralized to allow this as a general solution.

It is possible to model users as a community, to assign access rights over services to the community and to let the community administer the details. In this case, the service providers and the community leaders agree the definition of groups of users. The service providers assign access rights to particular groups, and the community manages the membership of the groups. When a new researcher joins a community, entry into the right groups, made once by the community administrators, grants access to all relevant services.

The 'OGSA security road map' for GGF specifies that there should be authorization servers and implies that authorization should be community-based. However, there are no standards, even in draft, for how an authorization server works in a grid.

The Community Authorization Server (CAS) from the Globus Project is a prototype of an authorization service. As yet, no production version has emerged and the prototypes are not suitable for production use.

Prototype grids tend to build their own authorization servers. EDG has the Virtual Organization Management System (VOMS). The e-Science data portal at CCLRC has an authorization server built in. AstroGrid has an access-policy/community system

designed against grid needs but not to grid standards. [PERMIS?...] These tools could be reused for the VObs.

I suggest that the VObs cannot, at this time, make a sensible choice amongst the various, partial systems. Nothing implemented now will be a complete and compatible solution in three years time. If we need grid-compatible authorization code, then we must wait for the GGF standard.

## Summary of recommendations

### **Web services**

- Continue to use web services as the basis of the VObs.
- Prefer SOAP services to simple HTTP-get operations as SOAP allows higher-level features like transactions to be added later.
- Exploit IT-industry standards and conforming products where they become appropriate. Prefer these to 'grid' products and standards if the two are incompatible.
- For VObs web-services that are in the public registry, prefer techniques such as WS-Context to OGSi.

### **OGSi and OGSA**

- Do not convert VObs services wholesale to OGSi; the gains are not great enough and the supporting software is not of sufficient quality.
- Do not use OGSi services in critical parts of the VObs (portal, workflow, etc.); they cannot easily be made sufficiently predictable.
- By all means exploit OGSi services written by third parties, but only at the edges of the tree of services. OGSA-standard services are preferable to arbitrary OGSi-compliant services as alternative implementations may be available.
- Avoid coding *services* in OGSi until better toolkits are available. Look for alternatives.
- Produce library code that makes coding *clients* for OGSi feasible.

### **Data-grid software**

- Use GridFTP as the main transport for files in the data grid.
- Do not use SRB as the basis of the data intergrid. It is OK in intragrids.
- Use OGSA-DAI (or ELDAS) to move relational tables. When the interfaces become sufficiently stable, consider reimplementing them independently of the grid products in conventional web services.
- Define VObs standards for data grid operations. Hide grid technology in the data grid behind a VObs façade.

## ***Access control***

- Use GSI-compatible X.509 certificates as the main way of identifying users within the VObs. (Portals can still use passwords at the system boundary.)
- Use WS-SecureConversation as the main authentication mechanism in the service grid. Use the WS-Security packaging for SOAP messages.
- Allow WS-Encryption, but deprecate it for normal services. I.e., clients must not expect an arbitrary service to support encryption. It is allowable for special applications.
- Open discussions, as a priority, on interoperation of identity certificates within the global VObs. Expect that full certification and interoperation will not be achieved for some years.
- Until VObs users have certificates, certify software agents to work on their behalf.