The Supplier Model for Legacy Applications in a Grid

Jonathan Giddy¹, Ian Grimstead¹, Jason Jones²
¹ Welsh e-Science Centre, Cardiff School of Computer Science, Cardiff University, UK
² School of Engineering, University of Wales, Swansea, UK
J.P.Giddy@wesc.ac.uk

Abstract

The use of web services as the basis for grid middleware has allowed scientists to wrap legacy applications as services in order to provide their capabilities to grid users. However, this “wrapper” model can be poor at satisfying the requirement for software and data to be efficiently positioned on processing resources around the grid as part of larger coordinated computations. We argue that some aspects of the earlier more exposed grid model, where the client coordinates the use of resources, support these issues better and allow more efficient grid solutions. We propose a model to take advantage of both systems.

1. Introduction

Computational grids provide the ability for users to obtain simple access to large amounts of processing power. There are now many examples of projects that have been made feasible through the use of computational grids. However, some projects which are technically similar to these successes have faltered due to issues surrounding licensing, ownership or access to resources.

In traditional distributed computing, resources are shared across administrative domains, but only after a heavyweight negotiation of the trust and access issues for each resource, perhaps accompanied by an exchange of paper contracts. Because of this barrier to obtaining access, distributed systems often consist of just enough resources to perform the desired computation, with minimal redundancy. The non-availability of any one component can cause the entire system to fail to perform the computation.

In comparison, grid computing aims to massively expand the scale of resource sharing to ensure the availability of adequate resources for any particular computation [3]. However, access to a larger number of resources necessitates a lightweight method of negotiating trust and access.

For this reason, grid developers have turned to a Service Oriented Architecture [2]. Service orientation allows a resource owner to provide limited access to specific functionality rather than full account-based access. Since the user has no access to the arbitrary capabilities of a user account, the degree of trust required between the user and the resource owner is lessened, and hence there is usually a lower overhead for gaining access to the service.

Despite the availability of such lightweight methods of access, many organisations are still using heavyweight account-based access to resources. Older grid middleware such as Globus Toolkit 2 (GT2), with its account-based access and mature support for the basic tasks of process management and file transfer, is well matched to the methods by which high-end computational providers have previously worked and they have little incentive to maintain an assortment of individual services.

Similarly, many providers of applications (and their users) continue with heavyweight negotiations because of the lack of support for licensing in systems that cross administrative domains [13]. Typically licence schemes are tied to a single administrative domain, inhibiting the sharing of the application across a grid.

In this paper, we examine the use of legacy applications (applications developed without reference to grid concepts) with both the “explicit” model of GT2-like middleware and the “wrapper” model of service oriented middleware. We propose an alternative “supplier” model for using legacy applications which provides benefits to the users and the resource owners.

This work has been inspired by a recent project with significant issues of application and data ownership and security [9], as well as several projects that did not proceed due to the complexity of commercial licensing issues.

In general, we consider the situation where a user wishes to transfer software and data to a computational resource to be processed. The various owners of the software, the data, and the computational resource all
want this processing to occur (either due to a direct interest in the result or due to indirect incentives such as payment). However each resource owner wishes to protect their own resource as much as possible.

2. The Explicit Model

We use the term Explicit Model to refer to the programming model typical of Globus Toolkit 2. In this model, the user explicitly names the resources to be used: the executable program file, the program’s arguments and environment, the data files, and the processing resource. The explicit model is shown in Figure 1. Note that the arrows in all diagrams indicate the initiation of action rather than data flow.

![Figure 1. The explicit model](image)

In this diagram the Client is the application that most directly represents the end-user. The Client is responsible for translating the user’s logical definition of the computation into an efficient physical computation. The Generic Processing Service exists on a computational resource in the grid and starts arbitrary processes for the user, although the service itself is provided by the owner of the computational resource.

The steps of the Explicit Model are:
1. The Client contacts a Processing Service, explicitly naming the application, data files and other attributes of the execution environment.
2. The Processing Service retrieves the application and data files from the user’s specified data resources.
3. The Processing Service starts the application.

The user explicitly specifies the location of the resource on which to start the process and the resources from which to retrieve the application and data files. Thus, the user is effectively responsible for using the middleware to grid-enable the application and for making decisions such as when to update the version or how to optimise the application for the processing resource.

3. The Wrapper Model

In the grid community, efforts to develop a Service Oriented Architecture have focused around Web Services based on WSDL, SOAP and UDDI [2]. Web Services provide a common language for specifying different services and the way they interact. This enables users to more easily combine services developed independently.

A common approach to grid-enabling legacy application is to provide a Web Service wrapper around the legacy application [6]. In the Wrapper Model, shown in Figure 2:

1. The Client contacts an Application Wrapper Service specifically developed as a front-end to the application. Each application has its own Wrapper Service.
2. The Wrapper Service retrieves the data files from the user’s specified data resources.
3. The Wrapper Service invokes the legacy application locally in an environment specified by the service provider.

![Figure 2. The wrapper model](image)

The application that performs the service is specifically hidden from the client to avoid dependencies between the client and service implementations. However, as shown in Step 2 of the model, it is still common for other resources to be explicitly named.

An advantage of the Wrapper Model over the Explicit Model is that there is no need for the user to have an account on the site running the application.
Authorisation to use the service can be controlled using techniques more scalable than requiring the compute owner to create accounts for each user.

In the Wrapper Model as shown, it is the compute owner who is responsible for grid-enabling the application and for deciding when to upgrade to a new version of the application or how to optimise the application for the processing platform.

This is because the model in Figure 2 has the wrapper service and the legacy application running on the same system. However, this limits the computational resources that can be used for a particular application to those that have the wrapper service and application installed. The providers of large computational resources, for example, high performance computing (HPC) centres, may not wish to support an unlimited number of services on their processing resources. Hence, some powerful resources may not be fully utilised in this model.

To overcome this, it is possible for a wrapper service to make use of the services that are exposed by computational resources. For example, many HPC centres provide the ability to securely transfer files and submit processes to a user account on their processing resource from a remote site (e.g. using Globus Toolkit 2).

![Figure 3. The remote wrapper model](image)

Figure 3 shows the wrapper model using a remote resource for the processing. In the remainder of this paper, we refer to this model as the Remote Wrapper Model. The steps of this model are:

1. The Client contacts the Application Wrapper Service.
2. The Wrapper Service builds a job specification explicitly specifying the application and data files, and sends it to the Generic Processing Service on a remote computational resource.
3. The Processing Service retrieves the application from a file repository owned by the application provider.
4. The Wrapper Service retrieves the data files from the user’s specified data resources.
5. The Processing Service starts the application.

Note that the Wrapper Service effectively takes on the position of the Client in the Explicit Model, except there are two user identities: the actual end-user (User A) providing access to the data resources, and the application owner (User B) providing access to the application resource. The Processing Service can run under either identity, but there are issues with each possibility.

Running the application in an account controlled by the end-user allows computational resources to correctly attribute work done to the correct user, and allows the user to keep their data files accessible only to themselves.

This approach is taken by GEMLCA [7]. GEMLCA uses delegation of credentials to allow the service to start a process as the client user. Although delegation is widely used in grid security [14], it is relatively immature, and complex uses may introduce a new set of security problems. For example, the ability to start a remote job using delegated credentials is disabled by default in the Globus Toolkit. GEMLCA requires the user to turn on this ability, potentially enabling the propagation of virus-like programs [10].

As an alternative, running the application in an account controlled by the application owner means the compute owners only need to create an account for the application owner, not for all the application wrapper service’s users.

XCAT [4] makes use of this approach. A problem with this approach is that the compute owner has no control over who is using their resource unless they setup a system for sharing usage information with the application owner (and with all other application owners).

Other problems exist in practice for using an application owner account. There is little incentive for an application owner to obtain access to large numbers of computational resources, and unless there is a large body of users demanding an application, there is little incentive for a compute owner to provide access to application owners. In addition, it is common for computational resource providers, particularly publicly-funded resources, to require a justification for the use of their resources. Typically, only an end-user can provide a suitable justification.
An advantage of the Remote Wrapper Model is that it separates the providers of the application and computational resources. The application owner can provide a fully-featured and up-to-date application service, while the compute owner can concentrate on providing a reliable well-scheduled computational resource. As in economics, specialization promotes more efficient use of resources.

4. The Supplier Model

We have seen that the Explicit Model gives control to the Client as to exactly which resources are used in a computation. This provides an advantage when the Client wishes to orchestrate multiple resources in an efficient manner. On the other hand, the Wrapper Model successfully hides many of the implementation details that the Client doesn’t care about (operating systems, processor types), but also some details the Client should care about (such as the location).

Consider a hybrid structure that allows the user to retain control over which resources are in use. We modify the Application Wrapper Service to become an Application Supplier Service. This service does not instantiate the application itself, but supplies the application on demand to a computational resource.

![Figure 4. The supplier model](image)

In the Supplier Model shown in Figure 4, the following steps occur:
1. The Client contacts the Application Supplier to request supply of an application. This step can involve complex negotiations including details of application parameters, user authorization, licensing, and cost. The Application Supplier records the association of the Client’s identity to the supply request and returns a handle. This handle contains the location and access token for the application resource.
2. The Client contacts the Processing Service on the desired computational resource, providing it with the Application Supplier’s handle.
3. The Processing Service contacts the Application Supplier, providing the handle and other metadata (see Sec. 5.2). The Application Supplier checks the authenticity and validity of the handle and returns an appropriate application.
4. The Processing Service retrieves the data files from the user’s specified data resources.
5. The Processing Service starts the application.

The striking thing about Figure 4 is how similar it is to Figure 3. However, this belies the significant possibilities produced by this change.

Firstly, note that delegation moves in a different way. Instead of the Client delegating privileges to the Service, the Service is providing the Client with a token for receiving additional services at some future time.

Obviously the Client is more responsible for deciding where actions happen. This is powerful because within the scope of a particular computation, the Client is the centralised component, the only component to have the global view of the computation. Gannon et.al [4] suggest that a wrapper service can determine from the information provided by the client the best combination of resources to use for the service. This assumes that the service has a global view of the entire computation, or can understand an arbitrarily complex description of the full computation provided by the client. However services are designed to be mere components of a greater system. Only the Client, which is in effect the top-level service, has the global view of the computation (and often the benefit of being purpose-built for the type of computation involved).

Workflow tools such as Triana [12] are an example of a sophisticated client with a complex global view of a computation. Passing information about the interdependent services within the workflow to a single service in order for it to make scheduling decisions would appear to be needlessly complex.

5. Analysis of the Supplier Model

We now examine some situations where the Supplier Model demonstrates its advantages. We also consider some potential problems with the Supplier Model, and suggest areas for further work.
5.1. Licensing

A major problem in grid computing is the lack of suitable licensing models [13]. Application owners want their application to be used, but do not wish to weaken their licensing models to support grid computing at the cost of licensing revenue.

In the Explicit Model of grid computing, the client transfers the software to the computational resource before execution, either once in a pre-processing stage, or many times immediately before each job is submitted. However, when licensing is involved, a single user may not wish to pay for licenses for multiple sites.

In the Wrapper Model the computational resource owner installs the software on their resource, and then allows access to clients through the grid middleware. However, a computational resource owner may not wish to pay for all the applications used by all the users of a grid.

Both of these models stretch or break the licensing models of typical software. The first approach because the licensee (the user) may not have the right to create the many copies of the software required or to transfer the software beyond a defined administrative domain. The second approach because the licensee (the processing resource owner) may not have the right to make the software available to many users or because the licensee may not have the right to make it available to users outside a defined administrative domain.

Both the Remote Wrapper Model and the Supplier Model allow an application owner to provide a service specific to their application without also having to provide computational resources. This enables the application owner to accurately measure application usage by each end-user of the application. However, the Supplier Model does not limit the number of processing resources on which the user may run the application.

Although it certainly makes per-use licensing a possibility, the ability to accurately measure application usage does not prevent users from purchasing their licences by more traditional subscription mechanisms, or even mixing methods, such as using a site licence for processing resources within their institution, and a per-use licence for external processing resources.

5.2. Deployment and configuration

In the Explicit and basic Wrapper models, it is the end-user and compute owner respectively that are responsible for deployment and maintenance of the application. When an application is setup by the compute owner or an end-user with an account on the compute resource, the application can be configured to make optimal use of its environment.

In the Supplier Model, as in the Remote Wrapper Model, the application owner becomes responsible for keeping the application up-to-date and optimising it for the platform. In the Remote Wrapper Model, the application provider can test deployment on the set of resources available to the Application Service. However, in the Supplier Model, the Application Supplier initially does not know where the application is to be deployed.

In Step 3 of the Supplier Model, the Processing Service sends additional metadata to the Application Supplier. This metadata describes the platform and enables the Application Supplier to provide a suitable executable.

In addition to the operating system and processor type, the number of processors available to the job may be used to select multiprocessor versions of the application, or versions may be available to take advantage of hardware accelerator cards.

Many legacy applications make assumptions about their operating environment, usually based on the prime assumption that the application will be installed once under the supervision of an experienced administrator. In the Supplier Model, the dynamic installation of at least part of the application violates this assumption.

However, there is significant work being done on deployment, both in individual projects [5], and within the GGF OGSA Execution Management Services. Application owners are in the best position to fix these issues, and the Supplier Model allows them to solve the problems without necessarily becoming grid users.

5.3. Updates

Retrieving the application from the Application Supplier Service naturally lends itself to automated updates of the software.

Of course, an updated version, while possibly being more correct, may produce unexpected and therefore undesired results. However, version requirements can trivially be made part of the negotiation between the User and the Application Supplier.

A problem of the Supplier Model as shown is that the application is retrieved each time it is to be executed. In order to ensure reasonable performance some caching of the application may be necessary. However, this competes with the desire to have a record of each execution of the application. A small download of a consumable licence key would ensure
the application supplier is notified of each execution with minimal overhead.

5.4. Co-ordination

An application may be instantiated as part of a larger computation. For example, a single application may be part of a more complex workflow. This is typically not taken into consideration with the Wrapper Model, where even the fact that a computation is being migrated is hidden from the user. Exposing the location of the computation to the Client allows better co-ordination of the different parts for potentially more optimal allocations.

For example:
1. Selecting the location to run a computation based on the location of the data can have a significant effect on the efficiency of the computation [11]. Consecutive computations may run on the same processing resource in order to avoid transferring a very large intermediate file between sites.
2. Processing may be limited to taking place within a particular subset of resources due to the sensitivity of the data (e.g. defence, financial or healthcare data).
3. The controlling client can establish direct connections to the applications in order to watch the output of one application or to pipeline the output to another application [8].

6. Security Considerations

Owners of application, data, and computational resources want to protect their resources from misuse by other users, including each other. In this section we examine the ability of resource owners to protect their resources using the Supplier Model.

6.1. Application security

The Supplier Model allows the application to be run anywhere the user has access, even if the application owner was not previously aware of the computational resource. There is no trust relationship between the application owner and the computational resource.

However, the ability to supply versions of the application optimised for the user’s task also provides an opportunity for the application owner to protect the application from misuse.

The supplied application can have a restricted parameter range, reduced features, time restrictions or watermarks. While the application remains suitable for the requested task, its value as a general purpose application is reduced.

The usefulness of this approach also depends on the amount of variation provided through parameters provided to the application supplier as opposed to input data files. The more data provided through application parameters, the greater the ability of the application supplier to limit the general use of the provided application. On the other hand, this also requires the user to reveal more information to the application supplier about the computation to be performed.

A difficulty with this approach is the expense of recompiling code for each instance of the application. It may be necessary to limit such variability to a small module. Alternatively, a cryptographic licence key encoding any limits provides a relatively cheap mechanism to ensure that any misuse is obviously deliberate.

6.2. Data security

The non-executable data is frequently the most valuable and sensitive resource in a computation. In the Supplier Model the application owner does not get direct access to the data files. However, the application will naturally have access to the data files. To prevent any transfer of the data to the application owner (or any other party outside the computational resource), the Generic Processing Service can use operating system tools such as firewalls to prevent the application sending any messages on the network.

There is also scope within the model for the Client and the Application Supplier to conspire to protect the data from the computational resource owner. The Client’s negotiation with the Application Supplier could include agreement on an encryption key, to be used to encrypt the data and to be built into the supplied application. This would protect against most trivial or opportunistic attacks, although a sophisticated attack could be mounted using decompilation.

Many potential users of the grid are restricted to keeping their data within their own organisation. Defence and healthcare organisations are prime examples of organisations that continue to process their most sensitive data internally.

Such organisations can still benefit from the grid by accessing some services externally. The Supplier Model provides a large degree of flexibility in importing applications into organisations. A user can process local data on an internal grid using a remotely sourced application.
6.3. Computational resource security

The only elevated permissions required on the computational resource are those that may be required by the Generic Processing Service in order for it to change to the correct user account for the user. This meets a common requirement at many sites for only mature well-tested software to operate with elevated privileges.

The main risk over running a standard job is that the application is supplied by an unknown third party. There is a trust relationship between the user and each resource owner, but no trust relationship between the resource owners. The computational resource owner may wish to take some steps to isolate this application from key parts of the system setup, using appropriate operating system mechanisms [1].

7. The Supplier Model for other resources

The basic principles of the Supplier Model can be extended to types of resources other than applications. A logical extension is to apply the model to non-executable data files.

In the preceding sections we have considered this data to be available through simple file transfers. In many cases this is appropriate as the data is owned and supplied by the user. However, making data available through a Supplier service presents some interesting possibilities.

For example, data format conversions between systems are typically done by converting into a standard format at the sender and converting again at the receiver, or alternatively by using a single “receiver makes right” conversion requiring the receiver to handle multiple incoming formats. However, the ability to negotiate with a Supplier before actually beginning a transfer leads to the possibility of “sender makes right”. Data resources are in a strong position to perform these conversions as they have the best knowledge of their internal format, and hence can provide the most efficient implementation for converting to other formats.

Other resources may be accessed through a Supplier Service. Figure 5 shows a grid where application, data and processing resources are all provided through Supplier services.

1. The Client contacts each Supplier service to negotiate access to the resource. By doing this simultaneously, the Client can choose the best combination of resources for the computation.
2. The Client contacts the Processing Service with an explicit specification of the application and data resources to use.
3. The Processing Service retrieves the application…
4. …and data files.
5. (not shown) The Processing Service starts the computation.

![Figure 5. A supplier based grid](image)

In Figure 4, the Supplier Model diagram, we note that Site B, where the Application Supplier is running, is the only site not to use an account controlled by the Client. Instead, the Client, as part of its negotiation with the Supplier Service, obtains a token to allow access to the Application resource.

Applying this principle to all the resources in the computation, as in Figure 5, allows all sites to provide their resources without using an account controlled by the Client. Correct attribution of the user responsible for the resource access is maintained since the tokens provided by each Supplier Service can be mapped back to the Client through the initial negotiations.

One possible use is for data replication. Many legacy systems do not support the use of replicated data sources. However, by coordinating data, application, and processing suppliers simultaneously, a client can determine the best combination of possible resources for a computation.

8. Conclusion

The classic Explicit Model of grid computing gives the user the freedom to determine which resources to combine for an efficient computation. However, the price of this is the need to handle the complex issues of availability and heterogeneity.

The Wrapper Model works well for legacy systems: applications that have been developed on a platform where it is not practical to move the software off that platform. However, the Wrapper model is poor at
satisfying the requirement for software and data to be efficiently positioned on processing resources around the grid as part of larger coordinated computations.

The Supplier Model attempts to allow the client to use their knowledge of the complete computation to coordinate efficient placement of resources while moving inessential complexity to the providers of the most appropriate resources. As an additional benefit, it provides a framework for negotiation allowing further developments in the coordination of resources, quality of service, licensing of resources, and ultimately charging models.

9. Acknowledgments

This work was partly supported by the Welsh Development Agency under the “Grid for Wales” Testbed contract.

10. References


