

School of Computing Science,
University of Newcastle upon Tyne



An Architecture for the Dynamic Deployment of Web Services on a Grid or the Internet

Paul Watson and Chris Fowler

Technical Report Series

CS-TR-890

February 2005

Copyright©2004 University of Newcastle upon Tyne
Published by the University of Newcastle upon Tyne,
School of Computing Science, Claremont Tower, Claremont Road,
Newcastle upon Tyne, NE1 7RU, UK.

An Architecture for the Dynamic Deployment of Web Services on a Grid or the Internet

Authors

Paul Watson, Chris Fowler

Abstract

This paper describes an architecture for dynamically deploying Web Services over a grid or the Internet. Distributed job scheduling systems are found at the heart of most grid computing infrastructures. They allow jobs (a combination of the code to be executed and (in many cases) the data on which it is to operate) to be created by clients and dynamically routed to available, remote computing resources for execution. In recent years, there has been a trend towards utilising Web Services to build grid and other distributed applications. An application is represented as a set of services that communicate through the exchange of messages. However, if the computational requirements of a service cannot be met by its hosting environment then a job must be created and sent to a distributed job scheduling system for execution on a suitable host. Therefore, application writers must deal with the complexity of managing two different types of computational entities: services and jobs. The Dynasoar project is investigating an alternative approach in which there are no jobs, but only services. A service can be dynamically deployed on an available host in order to utilise its computational power, if no existing deployments can meet the computational requirements. This is analogous to remote job scheduling, but offers the opportunity for improved performance as the cost of moving and deploying the service can be shared over the processing of many messages sent to it. This is achieved in a way that is completely invisible to the consumer of the service. A key architectural feature is a clear separation between *Web Service Providers*, who offer services to consumers by advertising endpoints for them, and *Host Providers*, who offer computational resources. Separating these two components and defining their interactions makes it possible for them to be distributed over a grid or the Internet, and managed by different organisations. This opens up the opportunity for interesting new organisational/business models for Web Service and Host Providers. These include allowing the author of a service to make it available to consumers without providing the computational capability to process requests sent to it. It also creates the possibility for market-places in which Host Providers offer capabilities at a particular cost, and the Web Service Provider makes a dynamic choice between them. The paper describes the architecture, outlines a set of usage scenarios and discusses some of the design issues, including the need to express and enforce trust policies for the three main parties (Consumers, Web Service Providers and Host Providers).

Status

Version 1

Table of Contents

1. Introduction	1
2. Introduction to Web Services	2
3. The Architecture	3
3.1. Allowing the Consumer to select the Host Provider	6
4. Example Scenarios.....	6
4.1. Scenario 1: Dynamic Outsourcing of Web Service Hosting.....	7
4.2. Scenario 2: Making available a Service without offering hosting.....	7
4.3. Scenario 3: A Grid supporting a community of researchers	8
4.4. Scenario 4: Moving Computation to Data	8
4.5. Optimisations being Investigated	9
4.5.1. Exploiting a SOAP Gateway at the Consumer	9
4.5.2. Sending messages directly to the Host.....	10
5. Managing Trust Relationships.....	10
6. Related Work	11
7. Conclusions	12
8. Acknowledgements	12
9. Bibliography	12

1. Introduction

Distributed job scheduling systems that can dynamically route client jobs to available, remote computing resources for execution are now widely available (e.g. Condor [1], Globus [2], SunGridEngine [3]), and found at the heart of most grid computing infrastructures. The job – a combination of code to be executed and (in many cases) the data on which it is to operate – is created by a client and given to distributed job scheduling systems that aim to route it to a suitable host which has the resources available to execute it.

In recent years, there has been a move towards utilising Web Services to build grid and other distributed applications [4-6]. An application is represented as a set of services that communicate through the exchange of messages, and there are now widely accepted standards for describing interfaces (WSDL [7]) and the transfer of those messages (SOAP [8]). Consequently, many grid and other distributed applications are now constructed by harnessing a set of services. However, if the computational requirements of a service cannot be met by its hosting environment then a job must be created and sent to a distributed job scheduling system for execution on a suitable host. The consequence is that application writers must deal with two types of computational entities: services and jobs.

This paper gives an overview of work in the Dynasoar project, exploring an alternative architectural approach that removes the concept of jobs, and allows application and service writers and consumers to remain wholly within the service-based framework. Here, the equivalent to a job is the combination of a service, and a message sent to it for processing. To enable this approach, the architecture allows a service to be dynamically deployed on an available host in order to utilise its computational power, if no existing service deployments can meet the computational requirements. A key architectural feature is to make a clear separation between *Web Service Providers*, who offer services to consumers by advertising endpoints for them, and *Host Providers*, who offer computational resources. Separating these two components and defining their interactions makes it possible for them to be distributed over a grid or the Internet, and managed by different organisations.

We believe that this approach has potentially three main advantages:

- a) it simplifies the writing of applications and services by allowing writers to remain entirely in a service-oriented framework, rather than within one in which they have to deal with both services and jobs.
- b) it can improve performance by retaining the deployment of the code (in this case the deployed service) on a host. This allows the deployment cost to be shared over the processing of many messages sent to the service. In contrast, because jobs represent self-contained, “one-off” executions, job schedulers lack the ability to share the cost of the movement and installation of code across multiple executions. This cost becomes particularly important when code is large, or execution requires a complex environment to be set up, as is the case with many scientific and business applications.
- c) it opens up opportunities for interesting new organisational/business models for Service Providers, and Host Providers. For example, it allows the author of a service to make it available to consumers without providing the computational capability to process requests to the service; instead, these can be routed to specialist, third party Host Providers, or even back to local resources at the consumer (client). It also allows the creation of market-places in which Host Providers offer capabilities at a particular cost, and the Web Service Provider makes a dynamic choice between them.

This paper gives an overview of investigations into the design and application of this generic infrastructure for the dynamic deployment of Web Services. It is structured as

follows. Section 2 gives a brief introduction to Web Services describing how they offer the opportunity to exploit the approach we propose. Section 3 describes the architecture of the generic service deployment infrastructure. Section 4 describes various scenarios for how it could be used, and then Section 5 introduces a trust model to address the security issues these raise. Section 6 relates the proposed infrastructure to other work while Section 7 offers conclusions and describes the current state of our work in building and evaluating a prototype system.

2. Introduction to Web Services

Web Services provide a way of building loosely-coupled distributed applications. As defined in [9], Web Services interact by exchanging messages in SOAP format [10], while the contracts for the message exchanges that implement those interactions are described via WSDL [7] interfaces.

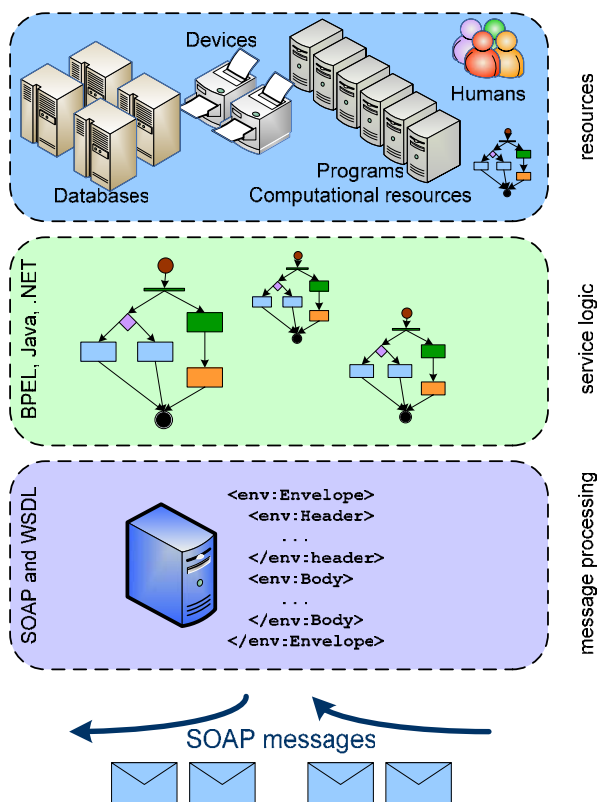


Figure 1. Anatomy of a Web Service

Figure 1 shows the logical structure of a Web Service. The provider of the Web Service informs potential Consumers (clients) of the address to which they should send SOAP messages for processing (the “endpoint”). This can be done, for example, by publishing the endpoint in a registry (e.g. UDDI [11]). When a message arrives at the endpoint, the service’s message processing logic transforms the network level SOAP message into a form appropriate for the service logic (often domain-specific objects). The message contents are then processed by the application logic, making use of the resources available to the service (e.g. files and databases). By encapsulating those internal resources within the service, and providing a layer of application logic between those resources and the consumers, the owner of the service is free to evolve its internal structure over time (for example to improve its performance or dependability), without making changes to the message exchange patterns that are used by service consumers. The separation between message handling, service logic and resources is an important property of services that we exploit in the infrastructure described in this paper as it allows us to freely decouple the service endpoint, to which messages are sent, from the location at which they are executed.

3. The Architecture

The aim of this work was to design a generic infrastructure for the dynamic deployment of web services, in response to the arrival of messages for them to process. This is achieved by dividing the handling of the messages sent to a service between two components – a *Web Service Provider* and a *Host Provider* – and defining a well defined interface through which they interact.

The *Web Service Provider* accepts the incoming SOAP message sent to the endpoint and is responsible for arranging for it to be processed. It does this by choosing a *Host Provider* and passing to it the SOAP message and possibly associated QoS information (as will be described below). It holds a copy of the service code (in a “Service Store”) ready for when dynamic service deployment is required.

The *Host Provider* controls computational resources (e.g. a cluster or a grid) on which services can be deployed, and messages to them processed. Therefore, it accepts the SOAP message from the *Web Service Provider* (along with any associated information) and is responsible for processing it and returning a response to the consumer (if one is required).

When the message reaches the *Host Provider*, there are two main classes of interactions, depending on whether or not the service is already deployed on the node on which the message from the consumer is to be processed:

Case 1: The Service is already deployed on the node on which the SOAP message is to be processed (to assist in the choice of node, it is likely that the *Host Provider* will maintain a registry of which services are deployed on which nodes). Here the *Host Provider* simply routes the SOAP message to a node on which the service is deployed and passes it to the local endpoint of the deployed service that will process it. Any response message is sent back to the consumer (currently this is routed via the *Web Service Provider*, but an implementation that used WS-Addressing could send it directly). This case is shown in Figure 2. A request for a service (s2) is sent by the Consumer to the endpoint at the *Web Service Provider* which passes it on to a *Host Provider*. The *Host Provider* already has the service s2 deployed (on nodes 1 and n in the diagram) and so, based on current loading information it chooses one (node n) and routes the request to it for processing. A response is then routed back to the consumer. Note that the *Web Service Provider* is not aware of the internal structure of the *Host Provider*, e.g. the nodes on which the service is deployed nor the node to which the message is sent. It simply communicates with the *Host Provider*, which manages its own internal resources.

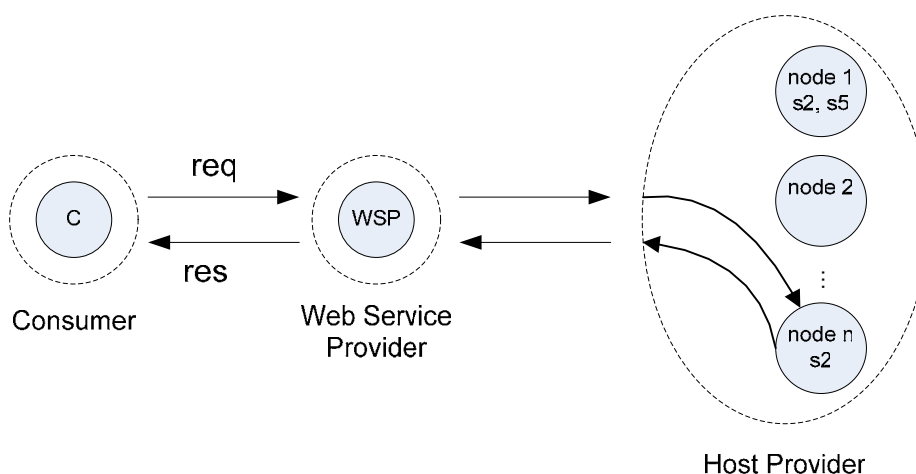


Figure 2. A request is routed to an existing deployment of the service

Case 2: The Service is **not** already deployed on the node on which the SOAP message is to be processed. This may be because the *Host Provider* has no nodes on which the service is deployed, or it may be because demand for the service is rising and so in order

to maintain acceptable performance levels the service must be deployed on another node.

To allow for the dynamic deployment of services, the Web Service Provider maintains a “Service Store” that holds deployable versions of services and associated metadata (in the prototype implementation, the Service Store is actually itself a Web Service, and so potentially it could be shared by several Web Service Providers). The architecture makes no assumptions about the nature of the deployable services except that, once deployed, they are Web Services that accept SOAP messages. In the simplest case, for Java-based services they may be files that are deployed by being copied into the appropriate directory of an Axis Tomcat server. However, for more complex cases they could, for example, be a set of executable commands that install a database, populate it by replicating the contents of another database, and then install a Web Services wrapper (e.g. OGSA-DAI [12]). Alternatively, they may be a “frozen” virtual machine containing a complex environment that can be copied to a host and installed there. Each service may have multiple entries in the Service Store, e.g. different versions for different types of host (e.g. Windows, Linux and Solaris).

Included in the information that the Web Service Provider sends to the Host Provider along with the SOAP message is an identifier for the service being called and the endpoint of the Service Store. If the Host Provider decides to deploy a service on a new node then it sends a message to the Service Store requesting the code for the service. When this returns, the Host provider installs it on the node and routes the message to its local endpoint. The message is then processed and the result (if any) returned to the consumer. An example of this case is shown in Figure 3. A request for a service (s2) is sent to by the consumer to the endpoint at the Web Service Provider which passes it on to a Host Provider (step 1 in the Figure). The Host Provider does not have the service s2 deployed on any of the nodes it controls and so, based on current loading information it chooses one (node 2), fetches the service code from the Web Service Provider and installs the service on that node (step 2). It then routes the request to it for processing (step 3). A response is then routed back to the consumer. Node 2 continues to host service s2, and so it is ready to process any other messages routed to it by the Host Provider.

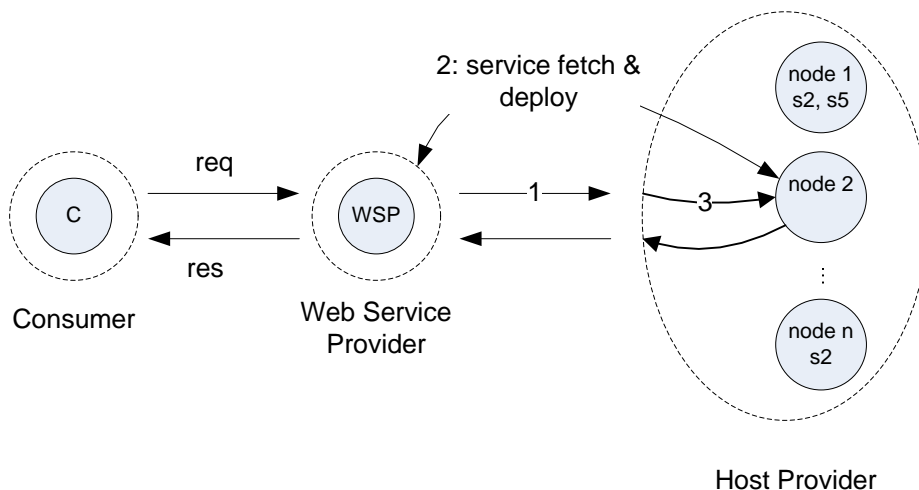


Figure 3. A service is dynamically deployed to process a request

It should be noted that the Consumer need not in any way be aware of the fact that a service is dynamically deployed – their interaction with the service is through the standard mechanism of sending a message to the service endpoint offered by the Web Service Provider.

Once a service is installed on a node it can remain there ready to process future messages until it needs to be reclaimed, perhaps to make way for other services that

have become more in-demand. This retention of deployed services has the potential to generate large efficiency gains when compared to job-based scheduling systems in which each job execution requires its own installation. A related project, the GridSHED project [13, 14] uses mathematical models to generate heuristics to determine when it is desirable to install a service on another node, versus when it is better just to use the existing installations.

In the above description, services are deployed in the Host Provider in order to meet changing demands, triggered by the arrival of messages from consumers. This does not however preclude pre-emptive service deployment by a Host Provider, for example where changing demands are predictable (e.g. regular peak times for a service).

In Figure 2 and Figure 3 only a single Host provider was shown. However, because there is a well defined interface for the interactions between the Web Service Provider and the Host provider, it is possible for the Web Service Provider to make a dynamic choice between Host Providers, for example on the basis of cost, availability of resources, or quality of service. This also opens the way for organisations or third parties to build marketplaces to match the requirements of the Web Service Providers against the offerings of the Host Providers (Figure 4). The Marketplace would offer the same interface as a Host Provider but, on receiving a message, would dynamically select a Host Provider that could best meet the needs of the Web Service Provider. It would then pass the request message on to the chosen Host Provider for processing (this would therefore be transparent to the Consumer or Web Service Provider). As before, if required the Host Provider could download a service from the Web Service Provider and dynamically deploy it.

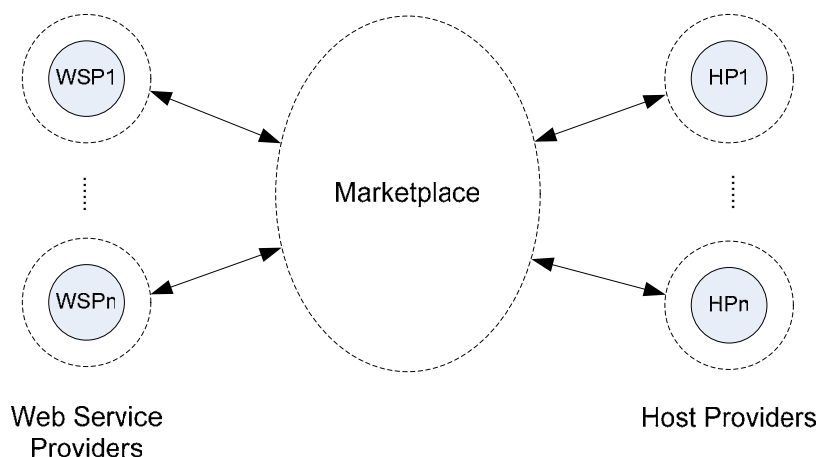


Figure 4. A Marketplace for Matching Web Service Providers to Host Providers

In implementing the Host Provider, the aim should be to build on the facilities provided by existing cluster and grid job scheduling solutions. Information management services (e.g. MDS [15]) allow loadings to be determined, so providing guidance on to which nodes messages should be routed, or new services deployed. At a higher level, the functionality of job scheduling systems can be used to identify hosts that already provide a particular service (e.g. using Condor [1]) and also a means to route messages to those nodes. Consequently, the Host Provider can be viewed as a high level service building on the functionality offered by lower level scheduling and information management services. For example, where the lower-level infrastructure is relatively sophisticated, QoS requirements could be provided by the Consumer in the header of the request and used in the selection of a node for processing the message.

The introduction of a market-place allows failure to be handled at three levels. Failure of a node in a Host Provider can be managed internally by directing messages to other nodes, and/or deploying the service on another node. Failure of an entire Host Provider can be handled by the marketplace or Web Service Provider selecting an alternate Host

Provider. Failure of a Web Service Provider can be handled through the normal Web Services mechanism of the Consumer going to a registry to find an alternative endpoint for the same service.

3.1. Allowing the Consumer to select the Host Provider

In some cases, it may be desirable, or absolutely necessary, for the consumer to specify the Host Provider, rather than leave this choice to the Web Service Provider. Examples of this are when:

- the Web Service Provider does not wish to deploy their own Host Provider, nor enter into a relationship with one. For example, a company may wish to sell access to their Web Services, perhaps by charging per message processed. However, they may not wish to be responsible for actually processing the messages sent to those services.
- the consumer has their own powerful compute resources that they wish to utilise. They therefore deploy their own, local Host Provider component. This may, for example, give them faster or cheaper hosting than they would get by using an external Host Provider.
- the consumer wishes to control the choice of host provider in order to exploit their knowledge of the behaviour of their application. For example, a consumer may know from experience that they get better performance from a service if it is deployed on a host that is close to a database on whose data the service operates.

These scenarios are described in more detail in the next Section. However, the solution is the same: Host Provider information is included in the header of the SOAP messages sent by the Consumer to the Web Service Provider. When the Web Service Provider receives the messages it uses this information to select the Host Provider specified by the consumer. For example, Figure 5 shows an example where a Consumer sends a message to a service offered by the Web Service Provider. The service is computationally expensive, but the consumer has access to local compute resources that is running the Host Provider software. Therefore, the consumer's tooling inserts the location of the local Host Provider in the header of the message sent to the service offered by the Web Service Provider. The provider then utilises that Host Provider to process the message. If the service is not already available on the specified Host Provider then it will be dynamically deployed in the normal manner.

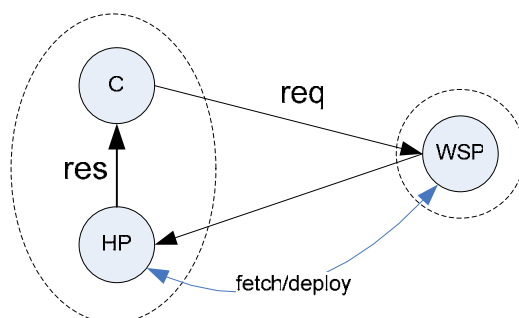


Figure 5. The Consumer Specifies a Local Host Provider

4. Example Scenarios

In this section we describe some common usage scenarios that can be supported by the infrastructure described in the previous section.

4.1. Scenario 1: Dynamic Outsourcing of Web Service Hosting

A bioinformatics company (BioCorp) wishes to earn revenue by writing computational services and charging customers to use them (perhaps by charging for each message processed). However, it does not wish to host the services. It therefore contacts a Web Services Hosting company (Hosting Inc) and enters into a business arrangement that will allow BioCorp's web services to be hosted by Hosting Inc (this is analogous to the way an ISP hosts web sites). To achieve this, BioCorp run the Web Service Provider component on their system, while Hosting Inc deploy the Host Provider component (Figure 6). When BioCorp produce a new Web Service, they advertise its endpoint to potential consumers and place a deployable version of it in their Service Store. When Hosting Inc need to install a service on one of their nodes in order to process a message, they retrieve it from the BioCorp code server, as described in Section 3.

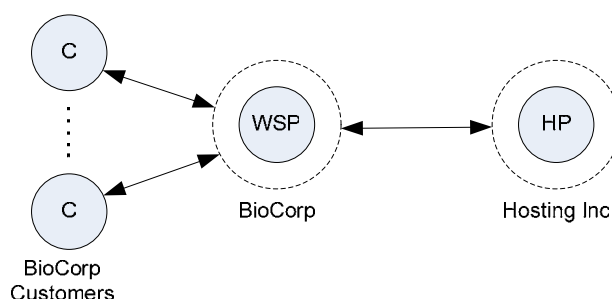


Figure 6. Scenario 1: BioCorp Dynamically Outsource to Hosting Inc

After some time, the CTO of BioCorp notices that other Web Services Hosting companies are available, supporting the standard Host Provider infrastructure, but sometimes offering hosting at lower cost. If the hosting companies offer an interface that exposes the current hosting cost (say per message, or per unit of CPU time, or per service deployment), and QoS capabilities then BioCorp can dynamically choose the cheapest host that meets any QoS requirements it may have. However, rather than manage this dynamic selection process themselves, if there is a 3rd party marketplace available (Figure 4) then BioCorp might find it easier just to utilise that.

4.2. Scenario 2: Making available a Service without offering hosting

A researcher has written a web service that they host on their own desktop computer, and use successfully for their own research. At a conference, they describe the service, which generates some interest and has others asking if they may use it. However, the researcher does not own the compute resources required to host access to the service by others - it is computationally intensive and so large CPU resources would be required to meet the requirements of the community of potential users. Nor could the researcher pay a Host Provider company to host the service as the researcher has no way of recovering the cost from users. Therefore, Scenario 1 is not appropriate.

The researcher could choose to make the code available for downloading on their website, but this would require potential users to be able to manually download and install it on suitable local resources. This requires a level of knowledge that many users do not have. Further, if the researcher upgraded the service, perhaps to fix a bug, then those who had already downloaded and installed it would not see the benefit.

Instead, the researcher utilises the solution described in Section 3.1 and shown in Figure 5. They deploy the Web Service Provider component locally and add the service into the Service Store. Any interested consumers must arrange their own Host Provider (perhaps by deploying it on one of their own machines, or by finding a Host Provider that they are allowed to use, perhaps one provided by their organisation). When the consumer sends a message to the endpoint, it sends in the header information on the location of the Host

Provider that can be used (this could be done by a client library). This allows the Web Service Provider to route the message to the Host Provider, which can, if necessary, retrieve the Service from the Service Store and install it as described in Section 3.

4.3. Scenario 3: A Grid supporting a community of researchers

This is an extension of Scenario 2. An organisation such as a university, corporate lab, or national body wishes to make a pool of computational resources available for researchers. They provide a grid infrastructure (e.g. a National Grid) that can accept and dynamically schedule jobs over a set of nodes. However, the researchers they support also wish to utilise computationally intensive services that they or others have written. Therefore, the Host Provider component is deployed on top of the grid. Researchers can then deploy the Web Service Provider infrastructure locally, and configure it to utilise the grid for dynamically deploying their services (Figure 7). This has the advantage that researchers can make their services available to colleagues, but leave the job of hosting them to a grid provider.

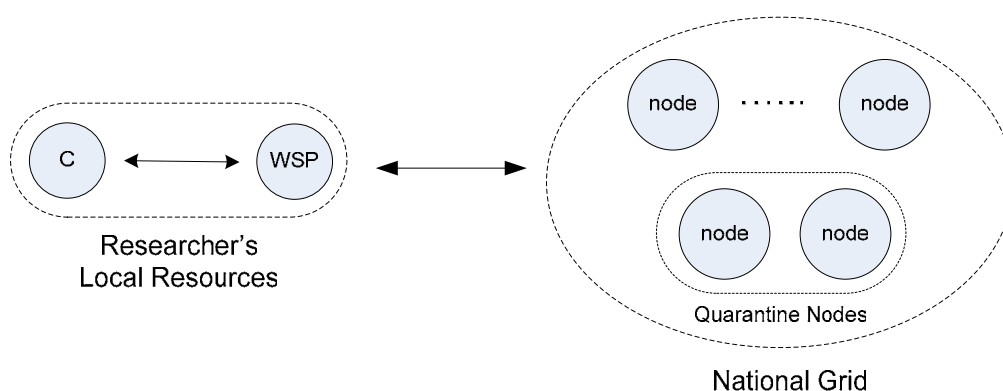


Figure 7. A National Grid Providing Resources to a Remote Researcher

The managers of the grid may be concerned about deploying arbitrary services on their nodes (though currently many grids accept arbitrary jobs from users). Therefore, they may choose to implement a policy such that when the Host Provider component receives a request to process a request for a new service, they deploy that service only on a subset of their resources (“quarantine nodes”) where its behaviour can be monitored to check that it will not disrupt other users. Once the managers are comfortable with the service then the restriction can be lifted, and the service would be allowed to be deployed on any suitable node on the grid. This could all be controlled automatically through the use of policies set in the Host Provider, which would specify any restrictions on the nodes on which a service could run. The policy for a service could be changed either automatically (as a result of monitoring the behaviour of a new service) or through manual intervention by a manager of the Grid.

This scenario still contains the danger that if a researcher makes available a service that proves popular, the researcher’s local Web Service Provider component will have to be powerful enough to accept all the incoming messages, and forward them to the Host Provider. To prevent this, the managers of the grid could also deploy a local instance of the Web Service Provider component on scalable resources (c.f. scalable implementations of a Web Server), and allow popular services to be made available (e.g. have published endpoints) directly from there.

4.4. Scenario 4: Moving Computation to Data

A bioinformatics service is written that analyses data from a database that is also made available as a web service [16] through an interface such as OGSA-DAI [12]. The analysis service sends a query to the database and receives back from it a large quantity of data.

It then processes this data and sends a small result set back to the consumer as shown in Figure 8 (the thickness of the arrows indicates the quantity of data transferred).



Figure 8. Interactions between an Analysis Service and a Database Service

In order to avoid transferring large amounts of data over long distances, it is advantageous to deploy the service as close as possible to the database on which it operates. The database provider realises that many services will benefit from deployment close to the database, and so provides an AIR architecture [17], with a cluster of compute servers close to the data (Figure 9). The cluster runs the Host Provider component. The provider of the Analysis Service (which may be a lone researcher or a commercial company that specialises in writing analysis services) runs the Web Service Provider component, and makes the service available through this. Consequently, either the consumer (through providing the information in the request header) or the Analysis service provider can specify that the service should run on the cluster close to the database service. Therefore when the Web Service Provider receives a message for the Analysis Service, it passes it to the cluster's Host Provider to ensure that it is processed close to the data (Figure 9:1). In the Figure, the Host Provider fetches and deploys the Analysis service from the Web Service Provider (Figure 9:2) before the request can be processed (on node 2). In the Figure, the Host Provider fetches and deploys the Analysis service from the Web Service Provider (Figure 9:2) before the request can be processed (on node 2).

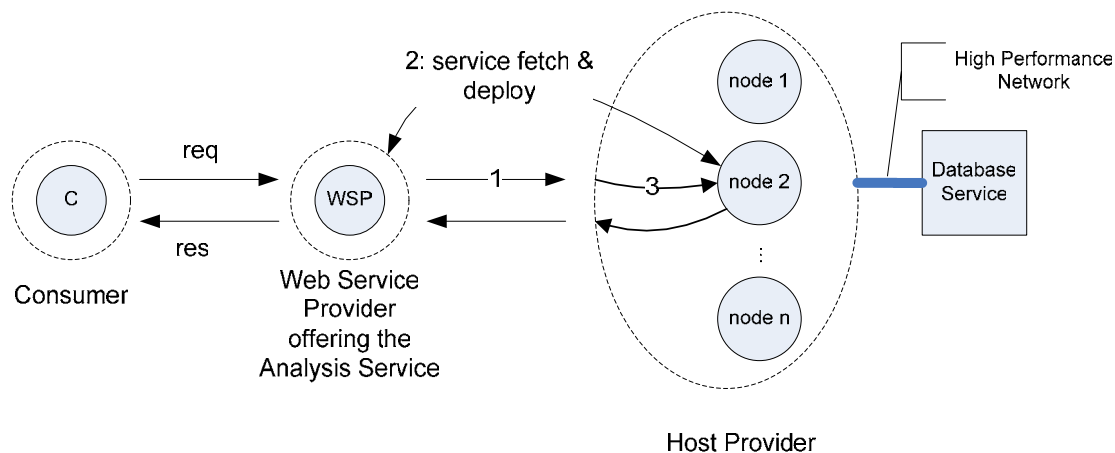


Figure 9. The Analysis Service is dynamically deployed close to the Database Service

4.5. Optimisations being Investigated

We are also investigating a set of optimisations that may prove suitable in some circumstances, though they raise issues in areas of security, charging and currency.

4.5.1. Exploiting a SOAP Gateway at the Consumer

The installation of a SOAP Gateway on the consumer opens up two new scenarios. The SOAP Gateway would act as an intermediary for all messages sent by a consumer. Whenever a service was deployed on the consumer's machine by the generic service deployment infrastructure then it would be registered within the gateway. Therefore, whenever a consumer sends a message, the local gateway could check whether or not a version of it was already deployed locally. If so then the message could be directly routed

to the service, so avoiding the messages from the Consumer to the Web Service Provider, and from the Web Service Provider to the host.

This has some potentially interesting advantages:

- a) fewer messages, lower latency
- b) the Web Service Provider does not ever see the consumer's message, and so this could be a way of dealing with applications where the consumer does not want the information in the message to travel outside its organisation. An example would be a pharmaceutical company that wanted to utilise a bioinformatics analysis service, but did not want messages sent to it to travel outside of their organisation, because they contained information that may give clues as to drug targets they were currently investigating.
- c) it would allow off-line access to a service. Once the service is deployed, it can be utilised even if the web service provider is not reachable, for example because the consumer is using a machine such as an off-line laptop or PDA.

However, there are also some potential disadvantages:

- a) the cached service could become out of date. e.g. a new version with a bug fix may not be utilised. This could be addressed by the Web Service Provider offering a mechanism by which hosts that deploy services could register to receive updates (or invalidations) automatically.
- b) if the Web Service Provider wishes to charge on a per call basis then this is more difficult to ensure if messages do not always travel through the Web Service Provider.
- c) The Web Service Provider loses control over which consumers are allowed to utilise the service (this issue is discussed further in the next Section).

Consequently, we feel the need to explore these options in more detail before considering them in real deployments.

4.5.2. Sending messages directly to the Host

An extension of the approach in Section 4.5.1 would be to re-architect the system such that the normal arrangement was that the Consumer sends the message directly to a Host Provider, and it is up to the host to contact the Web Service Provider to access and deploy the service, if this is required. This has advantages a and b above, but also disadvantages a, b and c. Consequently, these issues need further investigation before this approach is considered for real deployments.

5. Managing Trust Relationships

The dynamic deployment infrastructure described in this paper needs to respect the security and policy requirements of the three parties: the Consumer, the Web Service Provider and the Host Provider. To satisfy this, a Tripartite trust model is being explored (Figure 10). This has the aim of managing the relationships between all three parties using security and policy mechanisms.

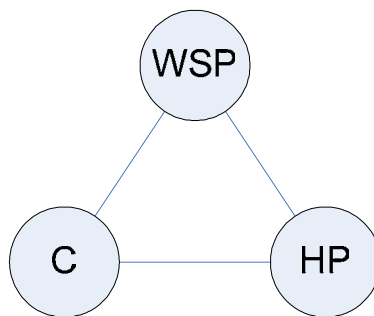


Figure 10. The Tripartite Trust Model

The relationships that the trust model must capture are:

- Web Service Provider – Host Provider. The Web Service Provider may wish to restrict the Host Providers it utilises. For example it may only wish to use Host Providers that it trusts to not offer the deployed services directly to consumers as a ploy to avoid paying the Web Service Provider for each use of the service.
- Host Provider – Web Service Provider. The Host Provider may only wish to deploy services from a restricted set of Web Service Providers. This may be to reduce the risks of hosting malicious code.
- Consumer – Host Provider and Consumer – Web Service Provider. The Consumer may wish to have their messages processed only by Providers they trust. An example would be a pharmaceutical company wishing to interact only with providers who are trusted not to monitor their requests and responses.
- Host Provider – Consumer and Web Service Provider – Consumer. The Providers may only wish to process messages from certain Consumers. For example, only those with whom they have a business relationship, so that they know they will receive payment.

We are currently exploring the use of Web Services standards and emerging specifications to allow these policies to be expressed and enforced (e.g. WS-Security [18], WS-Trust [19], WS-Policy [20], XACML [21]).

6. Related Work

The work described in this paper benefits from exploiting the results of related work that is producing components on which implementations can be built. In particular, the Host Provider can exploit work on remote job scheduling such as Condor [1] and Globus [2]. These provide ways to gather information on machine characteristics (type, software installed) and CPU loadings. Our aim is that the Host Provider can sit on top of existing grid infrastructure as a high level service.

However, the shift of focus from job scheduling to service deployment requires other issues to be addressed. A key is deciding when to deploy a service on a new node, rather than utilise an existing (though perhaps overloaded) deployment. This area has been extensively investigated in the GridSHED project for job scheduling [13, 14], and the results are now being utilised in the implementation of the Host Provider.

There are existing systems that are able to perform dynamic service deployment within a LAN. Of particular relevance is the work described in [22] which also includes a store for service code and dynamic deployment for loading and dependability reasons. However, a key difference is that this work does not address the separation of the Web Service Provider from the Host Provider such that they can be provided by independent parties, potentially distributed over a grid or the Internet. Our view is that making this separation,

and defining the interactions between the parties opens the opportunity for a range of new opportunities for grid and Internet Scale distributed computing, as described in Section 4. Therefore, the two projects have a different focus.

The architecture of the infrastructure is designed to be independent of the way in which the service is deployed. The initial prototype utilises the dynamic service deployment facilities of Axis/ Tomcat, but the intention is to add a range of other options including more sophisticated, emerging, distributed system deployment description mechanisms such as SmartFrog [23], and CDDL [24], once these have stabilised. We believe that Virtual Machines [25-27] are likely to play an important part in future dynamic deployment infrastructures through their ability to encapsulate specialised software environments. Further, they have the potential to offer a degree of hardware and operating system independence that will increase the range of deployment options for a particular service.

7. Conclusions

This paper has given an overview of work in the Dynasoar project on dynamically deploying services in response to changing demand and host availability. It does so by defining a generic infrastructure consisting of two components: the Web Service Host, and the Host Provider. By separating out these two components, and defining interfaces through which they can interact remotely, this approach offers opportunities to implement a range of different options for processing messages sent to Web Services on a grid or internet-scale distributed system. This includes opportunities to create a marketplace for Host providers spread over the internet. Further, it offers the possibility for specialist service writers to make available their services for others to use, without having to provide any hosting capability.

We are currently evaluating the behaviour of a Dynasoar prototype, in which the Host provider is built on top of Condor [1]. Messages arriving at the Host Provider are wrapped as Condor jobs, with the ClassAd used to ensure that the message is directed to a node that offers the required service (if none exists then the service is dynamically deployed before Condor is used to schedule the request). Building Host Providers as high-level services sitting on top of lower level grid infrastructure allows us to re-use the large investment that has been made in the design and deployment of distributed job scheduling systems. The aim is therefore to layer the Host Provider on top of existing grid deployments, rather than to require a completely new software infrastructure.

In parallel with this evaluation, the project is moving on to investigate some of the issues in more detail. Current work includes tuning the cost model [13] that governs when and where services are deployed; experimenting with services that require internal state (e.g. dynamically deploying replicas of databases to act as structured data caches), the co-deployment of sets of services (for example whole workflows), and extending the range of ways in which services can be deployed (e.g. Virtual Machines).

8. Acknowledgements

The authors would like to thank the following for many discussions that have contributed to this work: Charles Kubicek, Savas Parastatidis, Arijit Mukherjee, Mark Hewitt and John Colquhoun. Chris Fowler would like to acknowledge the support of the UK Engineering and Physical Sciences Research Council.

9. Bibliography

- [1] T. Tannenbaum, D. Wright, K. Miller, and M. Livny, "Condor - A Distributed Job Scheduler," in *Beowulf Cluster Computing with Linux*, T. Sterling, Ed.: The MIT Press, 2002.

An Architecture for the Dynamic Deployment of Web Services on a Grid or the Internet

- [2] I. Foster, C. Kesselman, and C. K. I. J. S. Globus: A Metacomputing Infrastructure Toolkit. I. Foster, "Globus: A Metacomputing Infrastructure Toolkit," *International Journal of Supercomputing*, vol. 11, pp. 115-128, 1997.
- [3] Sun, "Sun Grid Engine." <http://www.sun.com/software/gridware/>.
- [4] S. Chatterjee and J. Webber, *Developing Enterprise Web Services: An Architects Guide*: Penguin Books, 2003.
- [5] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, "Grid Services for Distributed System Integration," *IEEE Computer*, vol. 35, pp. 37-46, 2002.
- [6] S. Parastatidis, J. Webber, P. Watson, and T. Rischbeck, "A Grid Application Framework based on Web Services Specifications and Practices." <http://www.neresc.ac.uk/ws-gaf>, 2003.
- [7] W3C, "Web Services Description Language (WSDL)." <http://www.w3.org/2002/ws/desc>.
- [8] W3C, "SOAP 1.1." <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>.
- [9] W3C, "Web Services Architecture." <http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/>, 2004.
- [10] W3C, "SOAP Version 1.2 Part 1: Messaging Framework." <http://www.w3.org/TR/soap12-part1>.
- [11] OASIS, "Universal Description Discovery and Integration (UDDI)." <http://www.uddi.org/>.
- [12] OGSA-DAI, "OGSA-DAI." <http://www.ogsadai.org.uk/>.
- [13] J. Palmer and I. Mitrani, "Optimal Server Allocation in Reconfigurable Clusters with Multiple Job Types," presented at Computational Science and its Applications (ICCSA 2004), Assisi, Italy.
- [14] C. Kubicek, M. Fisher, P. McKee, and R. Smith, "Dynamic Allocation of Servers to Jobs in a Grid Hosting Environment," *BT Technology Journal*, vol. 22, pp. 251-260, 2004.
- [15] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman., "Grid Information Services for Distributed Resource Sharing," presented at Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10), 2001.
- [16] P. Watson, "Databases and the Grid," in *Grid Computing: Making the Global Infrastructure a Reality*, F. Berman, G. Fox, and A. J. G. Hey, Eds.: Wiley, 2002, pp. 1060.
- [17] P. Watson and P. Lee, "The NU-Grid Persistent Object Computation Server.," presented at 1st European Grid Workshop, Poznan, Poland, 2000.
- [18] OASIS, "Web Services Security (WS-Security)." <http://www.oasis-open.org/committees/wss>.
- [19] Nadalin A. (Editor), "WS-TRUST." <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-trust.asp>, 2004.
- [20] S. Bajaj, D. Box, D. Chappell, F. Curbera, G. Daniels, P. Hallam-Baker, M. Hondo, C. Kaler, D. Langworthy, A. Malhotra, A. Nadalin, N. Nagarathnam, M. Nottingham, H. Prafullchandra, C. v. Riegen, J. S. (Editor), C. Sharp, and J. Shewchuk, "Web Services Policy Framework (WS-Policy)." <http://msdn.microsoft.com/webservices/default.aspx?pull=/library/en-us/dnglobspec/html/ws-policy.asp>, 2004.
- [21] OASIS, "XACML." http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml.
- [22] M. Keidl, S. Seltzsam, and A. Kemper, "Reliable Web Service Execution and Deployment in Dynamic Environments," presented at Technologies for E-Services, Berlin, 2003.
- [23] HP, "The SmartFrog Reference Manual," 2004.
- [24] Global Grid Forum, "Configuration Description, Deployment, and Lifecycle Management. XML Configuration Description Language Specification," 8/9/2004 2004.
- [25] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Waeleld, "Proceedings of the Nineteenth ACM symposium on Operating Systems Principles," presented at Nineteenth ACM symposium on Operating Systems Principles, Bolton Landing, NY, USA, 2003.
- [26] VMWare, "VMWare." <http://www.vmware.com>.
- [27] Microsoft, "Virtual PC." <http://www.microsoft.com/windows/virtualpc/default.mspx>.