OGSA First Impressions - A Case Study Re-engineering a Scientific Application with the Open Grid Services Architecture*

Ben Butchart, Clovis Chapman and Wolfgang Emmerich
Department of Computer Science
University College London
Gower Street, London WC1E 6BT, UK
{B.Butchart|C.Chapman|W.Emmerich}@cs.ucl.ac.uk

Abstract

We present a case study of our experience re-engineering a scientific application using the Open Grid Services Architecture (OGSA), a new specification for developing Grid applications using web service technologies such as WSDL and SOAP. During the last decade, UCL's Chemistry department has developed a computational approach for predicting the crystal structures of small molecules. However, each search involves running large iterations of computationally expensive calculations and currently takes a few months to perform. Making use of early implementations of the OGSA specification we have wrapped the Fortran binaries into OGSI-compliant service interfaces to expose the existing scientific application as a set of loosely coupled web services. We show how the OGSA implementation facilitates the distribution of such applications across a large network, radically improving performance of the system through parallel CPU capacity, coordinated resource management and automation of the computational process. We discuss the difficulties that we encountered turning Fortran executables into OGSA services and delivering a robust, scalable system. One unusual aspect of our approach is the way we transfer input and output data for the Fortran codes. Instead of employing a file transfer service we transform the XML encoded data in the SOAP message to native file format, where possible using XSLT stylesheets. We also discuss a computational workflow service that enables users to distribute and manage parts of the computational process across different clusters and administrative domains. We examine how our experience re-engineering the polymorph prediction application led to this approach and to what extent our efforts have succeeded.

1 Introduction

We discuss our experience re-engineering a scientific application using the Open Grid Services Infrastructure (OGSI) [16], a new specification for developing Grid applications using web service technologies such as WSDL [4], SOAP [3] and WSIL [1]. Making use of early releases of Globus Toolkit 3 [14], the first implementation of the OGSI specification, we have wrapped Fortran binaries into OGSI-compliant service interfaces to expose the existing scientific application as a set of loosely coupled web services. We demonstrate how this re-engineering effort has improved the scalability, reusability and robustness of the application. We discuss how OGSI compliant middleware facilitated the development process and what the OGSI programming model offers in comparison to other Grid middleware solutions or more general distributed technologies. We evaluate what we have learnt from this exercise and how we intend to build on this experience to create a robust, reusable grid infrastructure to serve as a platform for a wide variety of resource hungry applications.

2 Motivation

The Open Grid Services Architecture (OGSA) [16] is an attempt to integrate web services technologies with existing grid standards so that developers of grid applications can benefit from broad commercial support for web services standards. Our aim is to road test this emerging technology at an early stage of its development using real scientific applications. We hope that putting OGSA through its paces with these applications will provide valuable feedback to the OGSA and grid community and promote the adoption of this technology by scientists.

3 Case Study

A computational method for predicting the crystal structures that could be adopted by a given organic molecule would be of great value to the molecular material industries. During the last decade, SL Price's group at UCL has developed a computational approach of searching for the global minimum in the lattice energy that is relatively successful for predicting the crystal structures of small, rigid, organic molecules [12, 2]. However the method is computationally intensive and the search described in [12, 2] took a few months to perform. Studies on larger molecules, which are more typical of manufactured organic materials such as pharmaceuticals, are not feasible without access to large grids or HPC resources.
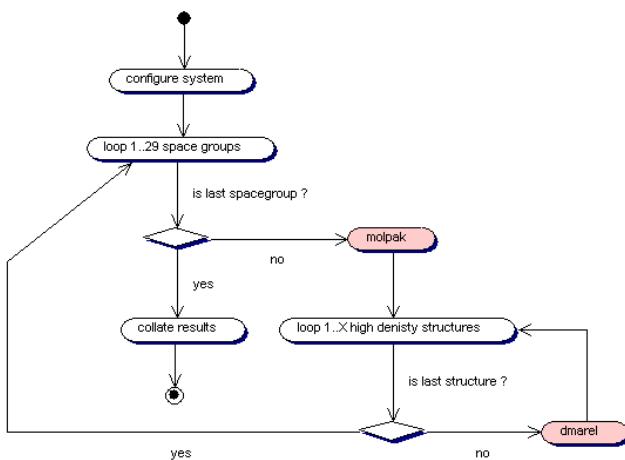

Figure 1. computational process of polymorph prediction application

Apart from the limitation of running code on a single processor, scientists must painstakingly check output files to determine whether particular results are valid and copy files from one machine to another so different parts of the process can run on platforms most suitable for a particular calculation. This lack of automation and resource coordination frequently results in errors forcing a total or partial rerun of the analysis.

The process developed by SL Price's group combines functionality from a number of different Fortran programs into a larger computational workflow illustrated in Figure 1. The highlighted activities represent the two main Fortran programs that have the most impact on processing time. Depending on the complexity of the input to these functions, a single computation can take anywhere from 5 to 90 minutes to complete on an Intel 686 Linux platform.

Two programs constitute a nested loop in the workflow, with the first program, `molpak`, undergoing 29 iterations

and the second program, `dmarel`, running 50 times for each `molpak` iteration. The only dependency between these computations is that the `dmarel` process cannot start until the `molpak` run has completed. Otherwise each of the 29 `molpak` computations and the 29*50 `dmarel` computations can run independently of one another. This raises the possibility of running `molpak` and `dmarel` computations in parallel across a large network of hosts radically improving performance of the system through parallel CPU capacity (Figure 2). Below we describe how we were able to achieve this by exposing the `molpak` and `dmarel` computations as distributed OGSI-compliant web services. Deploying this distributed version of the polymorph prediction application on a Beowulf cluster of 80 Intel 686 processors we were able to reduce the time taken for a scientist to perform an analysis on a compound such as aspirin from 3-4 months to around 20 hours. While we attribute some of this improvement to automation of the computational workflow we estimate that at least 50 days can be gained from running computations in parallel across nodes in the cluster.
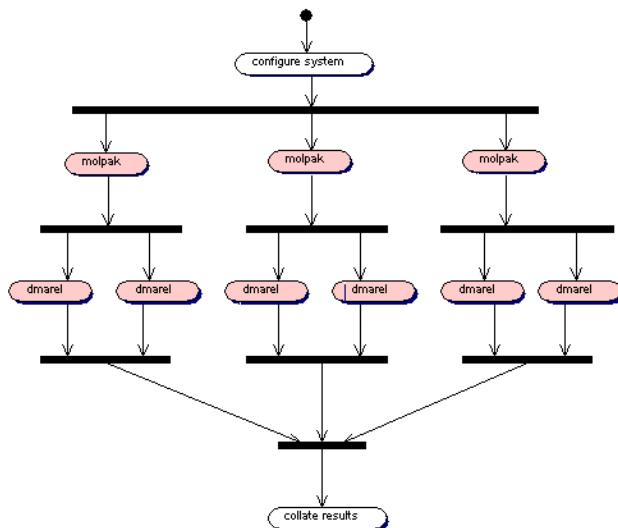
Figure 2. parallel model of polymorph prediction process

4 Re-Engineering Scientific Programs as OGSI Services

4.1 Why choose OGSA?

Web services technologies offer the Grid community a simple solution to heterogeneity and interoperability. The use of XML to provide both the description of service interfaces (WSDL) [4] and as a communication protocol be-

tween services (SOAP)[3] means that a service can be deployed on any platform and coded in any programming language that supports the web services model. Tools for automatic generation of client and server code greatly improve productivity and maintainability of applications. The OGSA specification augments these standard web services technologies with off the shelf reusable services for discovery and registration of services (Registry service), dynamic creation of service instances (Factory Service), asynchronous notification of service state changes (Notification Service) and resolving service addresses (Handle Resolver service). Altogether, OGSA compliant web services provide a powerful programming model for the development of grid applications.

But we recognize that most of this functionality is available already in various existing Grid or distributed object technologies. Mature Grid middleware toolkits such as Globus Toolkit 2 (GT2) [9] or CONDOR [15] offer registry, job management and notification services. There is no need for server and client stub creation since the various services conform to predefined interfaces, which have implementations supporting various OS and hardware platforms. So what exactly does OGSA offer us that we did not have before? The answer is a consistent and unified programming model for developing extensible services for Grid applications. The predefined services that GT2 provides are mostly independent of one another and have few if any common features. Since there is no common interface shared by all services, throwing together a system by combining individual services requires mastery of each API and effort enhancing one service is unlikely to improve or even work properly with other services. On the other hand, CONDOR offers a set of services that are very tightly coupled so that it is difficult to extend or replace individual components without completely reworking other components. OGSA offers developers a common framework for developing grid services that is both extensible and consistent.

The Grid service interface, from which all other services are built, provides common functionality and semantics that all services share. This common framework ensures compatibility between all OGSI compliant services making it much easier to aggregate and reuse services. Below we describe in detail how we were able to reuse core OGSA services in exactly this way to create our own higher level service infrastructure enabling us to deploy the existing polymorph prediction application across a large cluster of processors.

4.2 Architecture

Figure 3 shows the key components of the polymorph prediction application and interactions between them. Each rectangle represents an OGSI compliant web service and
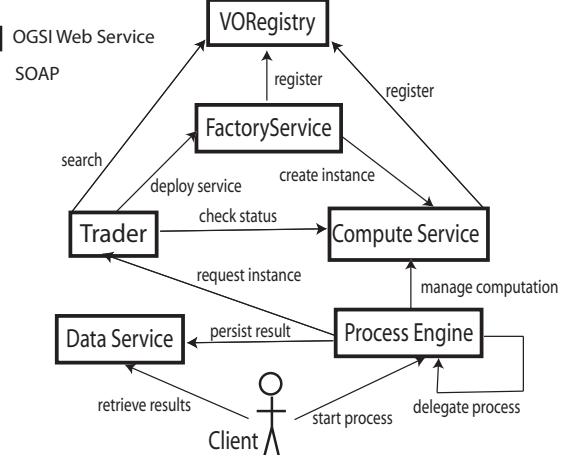
Figure 3. Service based e-science architecture

arrows correspond to SOAP requests with the arrowhead showing the direction of the request from the service caller to the service provider.

The client initiates a new process by sending a request to the process engine service. The process engine assumes responsibility for managing a computational workflow specified by the client. The process engine decides which compute service it needs to invoke next and contacts the trader service to get a reference to a corresponding service instance. The trader searches the VORegistry to obtain a set of references to instances of the required service type. It then checks the status of each service instance to determine whether the service instance is ready to perform a computation. If the trader finds an available service instance, a reference is returned to the process engine. If no service instance is available to run the computation, the trader asks the VORegistry for a factory service instance which it can use to create new instances of the target service. If no service instances are available and no new service instances can be created the trader sends a fault message back to the process engine, which waits for a certain amount of time before re-sending the request. Once the trader has returned a valid service reference the process engine invokes the compute service. The compute service executes the native Fortran binary on the resource. When the compute service returns the process engine triggers the next computation in the workflow or persists the result by invoking the data service. As well as invoking individual compute service instances the process engine may also delegate part of the workflow to another process engine instance.

Here we are using OGSI as a communication mechanism between distributed computations of a computational workflow. Instead of maintaining process relevant state in the

service instances we rely on the process engine to coordinate correlated messages between services. In this way we separate workflow logic and meta-data from the logic of computational components. This ensures that the computation service interfaces are purely concerned with discrete scientific functions and are not bound to a particular choreography.

4.3 Service Wrapping Legacy Code

Most Grid middleware platforms offer some kind of job management service that allow users to run existing binary codes such as Fortran programs as remote services across a set of networked resources. At a minimum we expect such Job Manager services to handle I/O streams, command line arguments and provide mechanisms for starting, terminating and monitoring jobs. Fortran codes usually use files to read input data and often use files as output. Most Grid platforms resolve this problem by providing some kind of file transfer system that enables users to copy input and output files to and from the particular resource where they are required.

There are a number of difficulties with this approach. First, transferring whole files across a network is wasteful since usually only a small part of the transferred data changes between requests. Our experience with the crystal polymorph application suggests that often 80% or so of the input file remains the same for each request, with maybe just a few keywords or data blocks changing between jobs. Results are hard to analyse in file format since data are rarely organised in a way that optimises searches. The distribution of relevant data across files makes aggregation and comparison of results inflexible. Applications that scientists write to analyse their results are not easy to reuse since they are tied too closely to particular file formats. Finally data are not represented in a consistent way with each application having a slightly different format for the same data structures.

The approach we have adopted is to expose the Fortran codes as web services that make the underlying binary codes completely transparent to the user. The interface of the binary codes including the input/output file formats, command line arguments and I/O streams is wrapped up in a web service interface that simply defines a set of operations and data structures. The wrapper is configured to use two conversion functions, one to convert the input data structures to the necessary input files and one to convert the output files to the specified output parameter.

Only non-redundant data is transferred across the network and users are not exposed to the full complexity of the underlying native interface and file formats. Also we represent data in a generic format. This means that we only need to write converters for the same data type once per application. Reuse of analysis tools is more likely as data is represented in the same way regardless of which application generated it. Extracting just the relevant data from output files allows users to store, search and aggregate their results far more efficiently facilitating use of database and data warehousing tools. By bundling a number of binaries into the same wrapper complexity of the interface can be hidden further (possibly at the expense of reusability) since a set of separate interfaces is presented to the user as a single operation.

One technique we have employed for converting input structures to the necessary Fortran input files is to intercept the SOAP message before it arrives at the Job Manager service and use an XSLT [5] processor to generate the input file from the body of the SOAP message. Unfortunately the mechanism for intercepting the SOAP message is a non standard feature of the Axis SOAP Engine [14] and is not portable to other Web Service platforms. The need to use such mechanisms highlights the restrictions imposed by the RPC messaging style. If the Job Manager instead used the document messaging style the SOAP header could be used to redirect the message to an XSLT service.

Overall we believe this wrapping strategy is a far more efficient, reusable and pragmatic approach to job management than solutions that expose users to the interfaces and file formats of the underlying binary code.

4.4 Deployment

The graphic below shows how we have deployed the polymorph application on a large Beowulf cluster (Figure 4). In the diagram we abstract deployment of services into three operational layers. The node layer at the bottom represents the private network and file system of a single system or cluster. These resources are not directly accessible to the outside world and can communicate only with other node layer services or with services operating in the domain layer. Domain layer services control access to node layer services and are responsible for the efficient utilisation and sharing of resources they control. External access to the domain layer is offered by services at the Grid layer that provide an interface to the outside world and coordinate work across different domains.

The service based architecture we have developed is flexible in the way it allows users to deploy a service at any operational layer. In the deployment shown above, for example, the computational process engine service is deployed at both the domain layer and the grid layer. But the constraints that the network topology imposes on the visibility of services means that these service instances play different roles on each layer. At the domain layer the process engine acts as a job broker, farming out individual computations to services operating on resources it administers in the node layer.

4
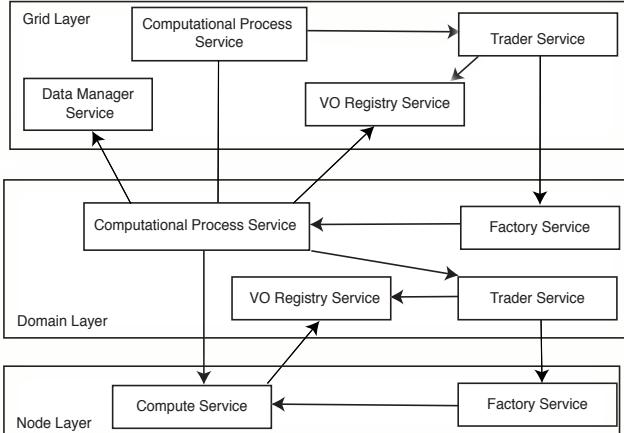
Figure 4. Deployment of service across three administrative tiers

At the Grid layer the process engine acts as a process broker delegating parts of the overall computational workflow to services deployed on the various domains it administers. We could go further and deploy a process engine service at the node layer, delegating part of the domain layer workflow to one of the nodes it administers, which would then farm out computations to a subset of its peers. This flexibility ensures that the overall system is scalable since responsibilities can be distributed more evenly across available resources.

5 Lessons Learned

The experience we have gained from deploying early implementations of the OGSA specification to re-engineer the polymorph prediction application has given us some insight into how OGSA works in practice and what areas we need to address to make development of OGSI-enabled Grid applications easier in the future.

A major technical difficulty we encountered was overloading of servers operating at the domain layer affecting services such as the process engine, the trader and VORegistry service. We found that our application did not scale well to large clusters consisting of 50+ processors. The problem arose from the tendency of individual events such as job completion to occur close together overloading coordinating services with too many requests at once. In the polymorph prediction process `dmarel` jobs are submitted in batches of 50 after each `molpak` run, leading to a spurt of simultaneous activity. This kind of behaviour is typical of real world computational workflows and demonstrates the importance of testing middleware on real applications. A short term solution to this problem was to upgrade and reconfigure services running on the domain layer and re-

deploy some of these services on a second system. Also we controlled job submissions so that only ten `dmarel` jobs were run for a particular `molpak` iteration at any one time. Although these measures solve the problem in the short term they do not provide a real solution to scalability of the application but merely expand the point at which performance starts to deteriorate. The real solution is to break up a large cluster into a set of smaller clusters by deploying coordinating services on resources in the node layer. To make this possible we need a mechanism for delegating parts of the computational workflow to other workflow managers. We are currently evaluating web service coordination specifications such as the Business Process Execution Language [7] to provide such a mechanism.

An area of difficulty we did not expect was exception handling. The crystal polymorph application generates a number of error conditions that we found surprisingly difficult to deal with. How error conditions are identified varies considerably. It could just mean checking an output file for a keyword such as "INVALID" or "SYSTEM FAILURE". The executable might itself throw an exception and return and error code that the job manager service will have to interpret. One situation that caused particular difficulty involved parsing an output file for keywords delimiting a particular dataset, analysing the dataset for certain conditions such as the presence of negative numbers (but only where the power was less than 4) and then repeating the job after incrementally removing all thus identified data elements in reverse order, not including the first and last elements in the dataset, which are ignored. Such complex rules are tricky and painful to implement in a language like Java that is not brilliant at parsing unstructured text. Worse still, the scientists often find it difficult to specify such rules clearly as they have not previously needed to automate and formalize their working practices. Indeed, the process of automating the system helped to improve the scientists' understanding of their own methodology. To emphasize this point it is worth mentioning that we have already changed the rule discussed above three times and are still not entirely sure that the algorithm is scientifically optimal. So our experience demonstrates the importance of systematically investigating what can go wrong, eliciting requirements from scientists and crucially testing whether the elicited requirements are in fact valid.

We were a little surprised to discover that the Core Service specification for OGSA does not include functionality for job scheduling and dynamic service deployment given how important these features are to any Grid application. Although many deployments will merely integrate with existing job scheduling systems we thought that some default job scheduler similar to our trader service would be provided. A simple interface that allows users to specify a registry containing schedulable service instances, define a

scheduling policy, establish user credentials and determine what service data is relevant to scheduling decisions would prove extremely useful. It would not be difficult to provide different implementations or extend such an interface to integrate existing resource management systems. This situation forced us to design and implement our own scheduler interface but with no guarantee that this will conform to other people's designs and implementations of OGSA.

Also missing are standardized procedures for deployment of services. There is no standard mechanism for transferring implementation code and native legacy codes, nor is there an archive structure and deployment descriptor specification such as exists for J2EE web and enterprise applications. We have had to write OS dependent scripts for deployment of services across the cluster, copying files to each node and restarting all servers. Given the concept of a virtual organization [10], it is reasonable to expect that a user can deploy applications without any knowledge of the topology or physical characteristics of the network. It really should be as easy as a J2EE deployment, where a set of class files and deployment descriptors are organized into a standard package structure, bundled into an archive, dropped into the `webapps` directory of a local server, thereby automatically replicated on all other servers in the cluster.

A further deployment issue that OGSA does not adequately address is the need for dynamic deployment of services. Although it is possible to deploy new service instances dynamically using corresponding factory service instances there is no way of dynamically deploying a factory service itself. Worse still, the deployment mechanism for such persistent services is platform specific. This means that the topology of nodes that can host a particular service remains static and is configured manually for each system. Although we can increase or decrease the number of instances running on this set of nodes to meet fluctuations in demand we cannot change the distribution of service capability across the total set nodes.

Early on we identified control of the computational workflow as an important area for development of Grid infrastructures. Our experience working with the crystal polymorph application shows that scientists often need to modify the computational workflow to support a particular analysis, sometimes adding a new computation or applying different computational strategies to the same analysis. So we need to enable scientists to change their workflows easily, preferably without assistance from software engineers.

The scientific application we have worked with demonstrates that fine-grain control of the computational process can have a big effect on efficient utilization of available resources. Most Grid middleware makes scheduling decisions at the single job (compute service) level but few consider the effect on the computational process as a whole. We believe also that delegation of computational workflow across ad-

ministrative domains would encourage more use of smaller clusters and reduce overuse of larger systems. In a situation where users have to break up the computational process manually to run across different systems users will naturally favour the larger and faster systems and ignore smaller systems which require the same effort to set up but offer relatively little additional compute resource. Also the effect of system failure is not as great since the overall computational process is more evenly distributed spreading the risk across several systems.

6   Related Work

The Globus Toolkit 3 (GT3) [14] has developed significantly since we began our work with preview releases. In particular many new high level services for job submission, data aggregation, file transfer and resource allocation have been built on top of the existing core OGSI services mentioned in this paper. The GT3 architecture is however different from the one we developed from the same set of core services. In particular, our approach to job submission and workflow differs from the GT3, where a complex computational process such as the polymorph application is generally submitted as a single job, having first copied input files to available resources using a file transfer service. The GT3 model is further complicated by vertical integration of authentication and authorization protocols, something we have not yet attempted in our prototype. The Globus toolkit now includes a deployment archive structure (GAR) that resolves some of the deployment issues discussed above, although this is something that really needs to be standardized in the OGSA specification.

7   Conclusions and Further Work

On the whole our first impressions of OGSA have been positive. Within a few months we were able to produce a usable prototype of the crystal polymorph application distributed across 80 machines, improving performance of the system through parallel CPU capacity, coordinated resource management and automation of the computational process. The exploitation of web services technologies contributed considerably to this success allowing us to quickly turn Fortran binaries into self-describing distributed services. We have been able to exploit the XML basis of web services through the use of XML conversion languages such as XSLT. We were able to develop our own scheduling service based on core OGSA services such as the Registry Service and Factory Service. Our future work will focus on exploiting emerging Web Services standards such as the Business Process Execution Language [7], Security Assertion Markup Language [8] to create a reusable and robust grid infrastructure.

Acknowledgements

Many thanks to Dr. Carole Ouvrard, Dr Graeme Day and Professor Sally Price of University College London Chemistry department.

References

[1] K. Ballinger, P . Brittenham, A. Malhotra, W. Nagy, and S. Pharies. *Web Services Inspection Language Specification (WS-Inpection) 1.0* . IBM and Microsoft, 2001. http://www.w3.org/TR/SOAP.

[2] T. Beyer, G. M. Day, and S.L. Price. The prediction, morphology and mechanical properties of the polymorphs of paracetamol. *The Journal of the American Chemical Society*, 123:5086–5094, 2001.

[3] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. Frystyk Nielsen, S. Thatte, and D. Winer. *Simple Object Access Protocol (SOAP) 1.1* . W3C, 2000. http://www.w3.org/TR/SOAP.

[4] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. *Web Services Description Language (WSDL) 1.1*. W3C, 2001. www.w3.org/TR/wsdl.

[5] J. Clark. *XSL Transformations*. W3C, 1999. http://www.w3.org/TR/1999/ REC-xslt-19991116 .

[6] J. Clark and S. DeRose. *XML Path Language (XPath)*. W3C, 1999. <http://www.w3.org/TR/1999/ REC-xpath-19991116>.

[7] F. Curbera, Y. Goland, J. Klein, F. Leymann, D. Roller, S. Thatte, and S. Weerawarana. *Business Process Execution Language for Web Services, Version 1.0.* BEA Systems, IBM, Microsoft, 2002. http://msdn.microsoft.com/library/ default.asp?url=/library/en-us/dnbiz2k2/html/ bpel1-0.asp.

[8] Farell et al. *Security Assertion Markup Language (SAML).* OASIS. http://www.oasis-open.org/ committees/security/docs/.

[9] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *Intl J. Supercomputer Applications*, 11(2):115–128, 1997.

[10] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*. The Globus Project, 2002. http://www.globus.org/ research/papers/ogsa.pdf.

[11] W. Hoschek. The Web Service Discovery Architecture. In *Proc. of the Int'l. IEEE/ACM Supercomputing Conference (SC 2002)*, 2002. http://edms.cern.ch/file/ 342747/1/wsda2002-1.pdf.

[12] W. D. S. Motherwell, H. L. Ammon, J. D. Dunitz, A. Dzyabchenko, P. Erk, A. Gavezzotti, D. W. M. Hofman, F. J. J. Leusen, J. P. M. Lommerse, W. T. M. Mooij, S. L. Price, H. Scheraga, M. U. Schmidt, B. P. van Eijck, P. Verwer, and D. E. Williams. Crystal structure prediction of small organic molecules: a second blind test. *Acta Cryst.*, B58:647–661, 2002.

[13] P. Prescod. *Slippery SOAP: The Two Sides of SOAP (scratchpad)*, 2003. http://www.prescod.net/soap/views/.

[14] T. Sandholm and J. Gawor. *Globus Toolkit 3 Core - A Grid Service Container Framework, Beta Draft*. The Globus Project, 2003. http://www-unix.globus.org/ toolkit/3.0beta/ogsa/docs/ gt3_core.pdf .

[15] T. D. Thain, T. Tannenbaum, and M. Livny. Condor and the Grid. In F. Berman, A.J.G. Hey, and G. Fox, editors, *Grid Computing: Making The Global Infrastructure a Reality*. John Wiley, 2003. http://media.wiley.com/ product_data/ excerpt/90/ 04708531/ 0470853190.pdf .

[16] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Marique, T. Sandholm, D. Snelling, and P. Vanderblit. *Open Grid Services Infrastructure*. The Globus Project, 2003. http://www-unix.globus.org/ toolkit/3.0beta/ogsa/docs/ specification.pdf.