

# Managing Scientific Processes on the *eMinerals* Mini-Grid using BPEL\*

Clovis Chapman<sup>1</sup>, Andrew M. Walker<sup>2</sup>, Mark Calleja<sup>3</sup>,  
Richard P. Bruin<sup>2</sup>, Martin T. Dove<sup>2</sup> and Wolfgang Emmerich<sup>1</sup>

<sup>1</sup> Dept. of Computer Science, University College London,  
Gower St, London WC1E 6BT, United Kingdom

<sup>2</sup> Dept. of Earth Sciences, University of Cambridge,  
Downing Street, Cambridge CB2 3EQ, United Kingdom

<sup>3</sup> Cambridge eScience Centre, Centre for Mathematical Sciences,  
University of Cambridge, Wilberforce Road, Cambridge CB3 0EW

## Abstract

The *eMinerals* mini-grid encapsulates a wide range of resources and services deployed across six sites in the United Kingdom. Scientists require means of exploiting these services in a fully integrated manner through the definition of computational processes specifying the sequence of tasks and services they require. The deployment of such processes, however, can prove difficult in networked environments, due to the presence of firewalls, software requirements and platform incompatibility. We propose here an architecture that builds on a *delegation* model by which scientist may rely on middle-tier services to orchestrate subsets of the processes on their behalf. For this purpose, we exploit the capabilities of the *Business Process Execution Language* (BPEL) standard, and other Web Service tools and standards, such as *GridSAM* and the *Job Submission Description Language* (JSDL). We define a set of inter-related workflows that correspond to basic patterns observed on the *eMinerals* mini-grid. These will enable scientists to incorporate job submission and monitoring, data storage and transfer management, and automated metadata harvesting in a single unified process, which they may control from their desktops.

## 1. Introduction

As grid infrastructures evolve and offer an increasingly important amount of resources and services, it becomes clear that scientists are suffering from physical and software restrictions not addressed by current-generation grid middleware and tools. Scientific processes, such as those defined by the scientists of the *eMinerals* project [1], involve a large number of services and resources, such as job execution services, data stores, visualisation services, etc., and potentially complex interactions between these. The realities of networked heterogeneous environments make the deployment of such processes an extremely difficult task: firewalls, platform incompatibility, software and resource requirements, security, etc. - all these elements can stop the average user from launching and coordinating complex computational processes from their desktops and/or applications of their choice. Ensuring that scientists can retain their current working environments and applications, with minimal or no change, is key to facilitating the transfer to grid environments and enabling the step change in the research that this

provides. Existing grid middleware can prove difficult to install, configure and use, and hardly provides the level of integration required to seamlessly incorporate a wide range of services into a unified process.

We have had the opportunity to work on the deployment of several scientific workflows on the *eMinerals* mini-grid [2] - a production level infrastructure encapsulating compute and data storage resources at several sites across the UK. While the scientific goals of these processes may differ, they present several similarities and common requirements. Specifically, they require batch job submission and monitoring primitives, the ability to store and retrieve large amounts of produced data and finally the ability to organize and index this data through the definition of corresponding metadata.

The approach that we adopt here is to decompose large scientific processes into a collection of basic patterns that can be fully automated and delegated to third party VO-wide systems, which will

---

\* Research presented has been funded by NERC through Grant reference numbers NER/T/S/2001/00855, NE/C515698/1 and NE/C515704/1 (*eMinerals*).

orchestrate the execution of these subsets of the process on the user's behalf.

We propose an architecture that builds on this delegation model, relying on the *Business Process Execution Language (BPEL)* [3] and other Web Service tools and standards, such as *GridSAM* and the *Job Submission Description Language (JSDL)* [4], to provide VO-wide orchestration and service provision. The increased adoption of Web Services by the Grid community makes the use of this industry-led standard in a Grid environment very appealing: BPEL is an orchestration language, enabling us to build services whose role is to coordinate interactions between Web Services according to specified workflows.

The architecture relies on the definition and deployment of a number of hierarchically organized workflows that correspond to basic patterns observed in our mini-grid. Scientists may trigger the execution of these workflows remotely through a *lightweight* self-contained client. These clients can be invoked by the user directly or within other applications as an external process, providing him with means of composing more complex workflows at a more abstract level; with applications, tools and languages of his choice – such as simple batch scripting. Alternatively, focus on re-usability ensures that users can incorporate our workflows into larger BPEL workflows.

We begin by covering some of the requirements of the scientists using the *eMinerals* mini-grid. We follow with the description of our architecture and a description of its implementation. Finally we demonstrate our use of it, through its integration into the *GTK Display Interface for Structures (GDIS)* – a graphical application for the display and manipulation of isolated molecules and periodic systems.

## 2. Deploying workflows on the eMinerals mini-grid

### 2.1 The eMinerals mini-grid infrastructure

Since 2002, we have been instrumental in building and maintaining the “*eMinerals mini-grid*” [2], which provides scientists of the *eMinerals* project a production level grid infrastructure to handle their large compute and data storage requirements. Spanning 6 sites in the UK, it incorporates a number of dedicated and shared data and compute resources. More specifically it provides the following compute resources:

- Three 16 node clusters (situated in Cambridge, UCL, and Bath).
- 2 IBM P-Series parallel computers (Reading).
- UCL Condor pool: Constituted of over 1000 student Windows Terminal Servers.
- Cambridge Campus Grid: Flocked Condor pools constituted of roughly 140 Linux nodes.
- Sunfire v800 Machine (UCL).
- Apple XServ 5 dual-CPU nodes cluster running Condor.
- 40 node PBS Cluster (Cambridge).

These resources all provide a *Globus Resource Allocation Manager (GRAM)* front end [5], as a standard interface to the various underlying resource managers.

For data storage requirements, we rely on the SDSC *Storage Request Broker (SRB)*, to provide seamless access to distributed data resources. We provide 5 SRB data storage vaults spread across 4 sites (UCL, Cambridge, Reading, Bath) amounting to a total capacity of roughly 3 Terabytes. Finally, we also provide a *metadata annotation service (RCommands service)*, hosted by the CCLRC, and database back end, in order to provide means for users to generate and search metadata for files stored in the SRB vaults.

Users will also regularly complement these resources with external resources such as those made available by the *National Grid Service* [6].

The large number of resources and their geographical distribution clearly highlights the difficulty in managing and providing access to such an infrastructure. The various sites are independently administrated and protected by institutional and/or departmental firewalls. Users also require a certain number of tools. At the very least we would expect users to have the Globus toolkit, Condor-G (for job scheduling capabilities), the RCommands client tools and finally SRB client tools for data storage and retrieval. Further more, even if we assume platform compatibility, these often rely on customized versions of mainstream libraries, and installation, configuration and maintenance can prove tedious, particularly where administrative access to the machine is not available. A critical assessment of the usability of current generation grid middleware and tools in the *eMinerals* environment has been offered by Marmier [6].

We aim here to solve two specific problems: firstly, we want to provide the user with means of taking advantage of these resources in a fully integrated and seamless manner, rather than as a

loose collection of individual services. Secondly, we want to ensure accessibility of our mini-grid by keeping client side constraints and requirements to a bare minimum.

## 2.2 Scientific workflow requirements

Scientific problems will require the definition of computational processes usually involving the execution of several data management and computational tasks. As a running example, we refer to previous work [7], where we tackled the problem of distributing the computations required to study the adsorption of pollutant molecules on a pyrophyllite surface. While the science behind the process may be outside the scope of the paper it is important to note the following: this is a highly parallelizable and repeatable workflow, which primarily involves over 200 executions of SIESTA [9] – and requires the additional use of external libraries to generate the molecular structures for the input, and means of extracting and composing the output as part of the process.

**Job specification and submission to grid clusters:** Though local input files and composition of output files may be handled by the user's local machine, the compute requirements of SIESTA required their parallel execution on available grid resources, with individual run-times of roughly two hours on our PBS clusters.

Whilst the Globus GRAM may provide standard job submission and monitoring primitives, clients require means of delegating the scheduling and monitoring of jobs to third parties. A successful job execution requires, beyond the basic specification of the requirements of the job (application type, executable file, input data, etc.), the selection of a suitable resource, the staging of the required data, regular polling of the job state (or the ability to receive state change notifications), and finally retrieval of the produced data. If the job failed to complete, fault handling may involve automated re-submission, migration of jobs, or if it could not be recovered, for example in the cases of erroneous job specifications, clean up of the remote execution environment and some form of roll-back. The management of such a submission is best left to third party servers with sufficient resources, and ensured access to the services required. It may not always be possible for long running workflows to be monitored continuously from certain user devices such as laptops.

**Data management:** Jobs and complete workflows can produce very large quantities of data; often several gigabytes in size. Though the largest output

files for our example rotate around 20 Megabytes, each execution of SIESTA can output a total of 111 output files and requires 7 input files.

Because of the large amounts produced, and because scientists may wish to share the data produced, we rely on the SRB to provide the space and robustness required, and generally serve as a data staging environment independent of the resources. However, file transfers between resources and/or client machines need to be carefully managed and integrated into the workflow. We must aim to minimise file transfers, whilst taking into account potential client-side limitations such as bandwidth limitations, unidirectional communication, etc.

**Automated metadata harvesting:** In addition, we also must stress the importance of metadata harvesting. The data produced must be categorized, indexed and generally annotated to ensure data re-use and data sharing between scientists, and this requires clear and searchable metadata. Automating the collection of metadata is most sensible for several reasons: firstly, it frees the scientist from the burden of annotating the data manually. Secondly, this ensures that a strict ontology is adhered to, facilitating future searches. Finally, much, if not all, of the metadata can be derived automatically from the specification of the job, such as date and time, user, application and resources used, etc. or, from the data itself.

A lot of research has been invested in the *eMinerals* project in solving the problem of software interoperability and data exchange. We have adopted the XML-based *Chemical Markup Language* (CML) [8] as a common data representation of chemical information, and worked on the adaptation of applications like SIESTA and GULP [10] to support CML data. The vast availability of XML processing tools also provides us with means to automatically capture data of interest, for metadata annotation purposes. In this way, dynamically parsing the CML data should enable us to retrieve various parameters such as molecule types, temperature and other parameters of interest, based on user requirements.

**Workflow decomposition:** From these requirements we can derive the fact that grid interactions of the scientific workflows of our users can be decomposed into basic units. For each individual job, we require the selection of a target compute resource, the retrieval and staging of data from the SRB onto the target resource, the submission and execution of the job, the storage of the produced data and corresponding metadata and

finally the clean up of the targeted resource, all of which constituting in itself a re-usable workflow. It should also be possible to interleave these grid-based units with desktop processes, such as retrieving user input for steering purposes, or input and output processing as per our example.

For the purpose of this work, we qualify a collection of single cluster parallel jobs, such as an MPI job, as an individual job.

### 2.3 The Business Process Execution Language and Web Services

Service-oriented architectures and Web Services particularly, are increasingly adopted by the Grid community as a standard and flexible means of specifying and developing services enabling the use of resources across autonomous domains.

Much work has been invested in the development of Web Service compliant grid middleware. This includes the fourth incarnation of the Globus Toolkit, through its implementation of the WSRF specification [5]. We have also previously participated in the integration of Web Service capabilities into the Condor job scheduling and resource management system [11].

The ability to combine these various services into a single integrated system is key to ensuring their collective use by the scientist. The Business Process Execution Language is designed to consolidate earlier work such as *XLANG* and the *Web Service Flow language*, into a single specification. It aims to enable the orchestration of Web Services, by providing means of composing a collection of Web Service invocations into an executable workflow, itself presented as an invocable Web Service.

The BPEL language allows us to specify the process to execute upon request as an XML document. It provides support for controlled flow elements, including sequential or parallel executions, conditional executions, variables and scoping. Advanced capabilities include support for XPath and XSLT and fault handling elements.

There are numerous advantages to this orchestration approach:

- *Development ease*: BPEL provides a “programming in the large” approach to workflow definition, enabling us to focus on high-level interactions between services. Graphical editors are also available, and we rely here on the BPEL graphical being developed at UCL as part of an OMII funded project [12].
- *Thin client*: A BPEL engine will be responsible for orchestrating the workflow on a user’s

behalf, acting as a middleman between resources and client.

- *Administration*: The BPEL workflow can be modified independently from the users, considerably easing administration and enabling the transparent addition of new resources and services.
- *Software availability*: Several implementations of the BPEL standard are available, which also provide monitoring and administrative tools.
- *Firewall management*: Due to the fact that SOAP calls can be embedded into HTTP, Web Services enable multiple sessions or services to be managed over a single port. HTTP outbound ports are often left open for web browsing purposes.

## 3. Architecture

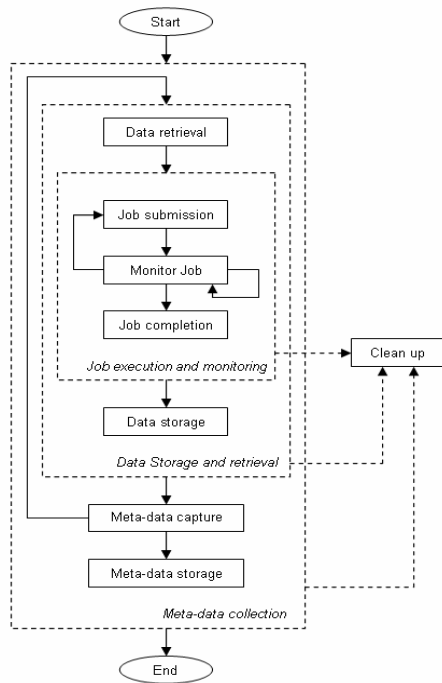
The architecture that we propose here is centered around one or more VO-wide BPEL engines, responsible for managing the processes required on behalf of the user, and coordinating service interactions. The result is a four-tier architecture as illustrated in figure 2.

### 3.1 Basic Patterns

We begin by defining a number of inter-related patterns that match the basic requirements of our users. These are hierarchically organized, as they are co-dependent, as presented in figure 1:

- *Job execution and monitoring pattern*: The most basic workflow pattern, it presents the ability to submit a job to a resource, monitor its execution until completion through regular polling, and handle any potential job faults (which may include resubmitting the job if needed).
- *Data retrieval and storage pattern*: Adds to the above the capability of retrieving data from the data vaults before job submission and uploading results to the vault upon completion.
- *Metadata annotation*: Generating metadata for the created file, and uploading these to the metadata server with mappings to the physical location of the data. Identifying elements of interest can enable the dynamic generation of new data and re-submissions.

This hierarchical organization maps well to BPEL-based workflows. Because these are themselves presented as Web Services, it is possible for a workflow to incorporate other workflows into its structure. Not all capabilities presented, such as metadata capture, may be required for every job of



**Figure 1:** Hierarchical workflow patterns

the scientific process that the user wishes to execute, and this flexibility is necessary to avoid unnecessary transfers or metadata. This hierarchical approach also favours re-usability and enables the potential addition of new services and its incorporation into larger workflows.

### 3.2 Architecture

We discuss here some aspects of how our delegation approach can be incorporated in an existing grid infrastructure. We can decompose the various areas of functionality as illustrated in figure 2:

**Application layer:** The interface presented to the user should be that of a local self-contained executable service, through which he can specify a job specification and data handling issues and which he can integrate in his own workflows and applications.

**Orchestration layer:** The orchestration service is responsible for handling workflow patterns on behalf of a client, as defined in section 3.1. It must present an interface to the client that is sufficiently flexible to allow a wide range of jobs or data transfers to take place. For this purpose we adopt the XML based *Job Submission Description Language (JSDL)*, an emerging GGF standard [13], enabling the specification of jobs and

accompanying data transfer requirements, though we extend it with additional environment specific characteristics, such as metadata parameter selection, or SRB specific details.

**Service abstraction layer:** These services facilitate higher-level interactions with the underlying resource services. These include:

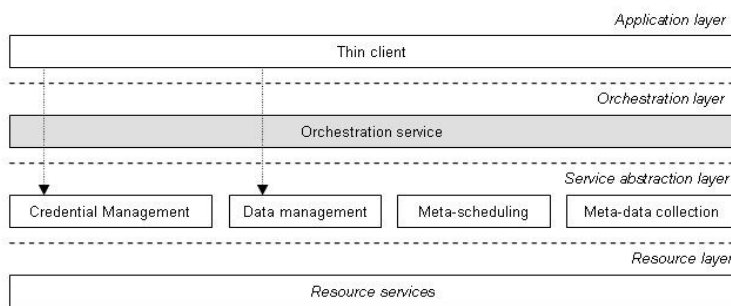
- *Metascheduling service:* The metascheduling service will be responsible for the distribution of the job to various underlying resource management systems, as well as the scheduling of jobs across resources. It must provide a job submission and monitoring interface.
- *Data management service:* This service will enable third party orchestrate of data transfers between independent resources. This may be achieved in multiple ways, though the interface to the service itself must present transfer operations between source and target locations.
- *Metadata collection service:* This service is responsible for the collection of metadata for a given data resource, and its storage. It must present an interface through which a client may select data elements of interest to capture at runtime, using a query language such as XPath, or as we do here, using the AgentX framework and RDF [14].
- *Credential management service:* An important notion to discuss is that of trust. Though the user may trust the orchestration service to act on his behalf, he might not necessarily be aware of all the parties involved in the process, which can be problematic particularly in cases of services requiring certificate proxy delegations. We aim to ensure that the user does retain some level of control by introducing an independent credential management service, which may be selected by the user and which will be responsible for authenticating each service requiring the user's proxy according to the user's specifications.

**Resource layer:** The resource layer represents all the lower level services enabling the use of the underlying resources. This includes job submission services for individual clusters, such as the Globus GRAM, the SRB vaults, metadata database services, etc.

## 4. Implementation

### 4.1 BPEL implementation and deployment

Our implementation of the architecture presented above is illustrated in figure 3.



**Figure 2:** Layered Architecture

**Client side tools:** As we have adopted JSDL as our main job description language, our client must provide means of generating JSDL and data requirement specifications.

We favor here a simple command line-based interface, which will allow users to specify the various requirements of the job (input files, executables, etc.) as a sequence of arguments, as well as allowing additional attributes (proxy server descriptions etc.) to be obtained from a configuration file, in a pre-defined location or from local environment settings, as well as support for standard streams (input, output, error).

The client also provides additional data staging capabilities. Because client side inbound connections are rarely available, it enables users to upload required files to the SRB vaults before job submission if required through HTTP - as we will explore below, as well as making files to be made available through a variety of means (FTP, HTTP). Files can also be retrieved to the user's desktop upon job completion. Finally, it also allows users to generate a certificate proxy, which will be uploaded to a credential management service of the user's choice. Once data and proxy requirements have been handled, the client will invoke the orchestration services to orchestrate the execution of the workflow.

The client has been implemented in Java, ensuring that it will be compatible with most platforms, using the CoG kit [15], Apache Axis [16], Apache HttpClient and other libraries.

**Orchestration service:** We rely on the open source Active BPEL engine to deploy workflows as specified in our grid environment. The Active BPEL engine also provides graphical Web based monitoring tools, providing a detailed view of the execution process, which will ensure that users can check on the progress of their jobs. The main advantage of such an approach to monitoring is that we keep the client side tools as "light" as

possible, while ensuring they still have an interface powerful enough to investigate any potential issues that may occur.

**Credential management:** We rely on the widely adopted MyProxy [17] server, which enables the upload of certificate proxies, provides support for their renewal, and authentication of services wishing to act on the user's behalf.

**Metascheduling:** We rely here on previous work in which we created a

plug-in for the GridSAM job submission service to support submissions to multiple Condor pools, or Globus managed clusters through Condor's Web Service interface [11], effectively transforming gridSAM into a complete metascheduling service. We also provided support for predictive scheduling; using Kalman Filter Prediction to determine future utilization patterns on each cluster, resulting in considerable reductions to the global scheduling over-head and queuing times.

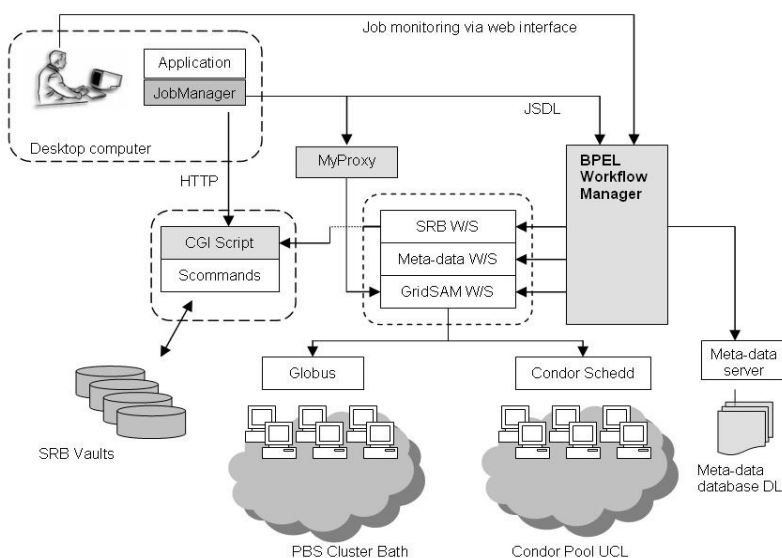
GridSAM also provides support for JSDL, and myProxy, facilitating its integration into the system, and required monitoring primitives.

**Data Transfer and services:** We require here means of coordinating transfers between third parties and our SRB storage vaults. While Web Services are used to control the data transfers, the actual transfer of data is handled through HTTP, in order to avoid the overhead of XML SOAP based invocations.

For this purpose, we have written a front end wrapper over the SRB tools using CGI that facilitates interaction with the SRB data vaults through HTTP, enabling the upload or download of files from the SRB using respectively HTTP POST and GET requests, with file, directory, username and password passed as request parameters.

Alongside this, we have created a Web Service that will facilitate the remote coordination of transfers. It is intended to be deployed on remote resources, and provides a simple interface through which we can request the upload of local files to the SRB vaults, or the retrieval of files into a specified location. It is also responsible for managing local file system resources, creating local temporary file sandboxes and ensuring the removal of unnecessary files upon job completion.

The reasons for adopting HTTP are obvious: they facilitate the upload of files for clients in the presence of firewalls, and also enable them to access files through their browser. It should be



**Figure 3:** Implementation

noted that SRB error codes have all been mapped to equivalent HTTP errors, to facilitate their automated handling.

**Metadata collection:** The interface provided by the RCommands Web Service provides means of adding, editing and removing new data sets and studies associated with a particular file, and its physical location (on the SRB).

Much of the data can be directly obtained from the basic job description, by isolating JSDL elements of interest, as well as additional notes provided by the user using the client side tool. However much of the detailed information about the data will be obtained by parsing the output files, particularly in XML or CML. We have created an additional metadata collection Web Service that will be responsible for identifying elements of interest given an appropriate RDF file and a suitably rich ontology, relying on the AgentX framework. These elements will then be passed on to the RCommands services by the orchestration service.

#### 4.2 GTK Display Interface for Structures

Allowing existing desktop application, whether graphical or command-line based, to transparently fork computations with heavier resource requirements to grid resources is a prime example of the capabilities required by the scientists.

As a use case for our work, and to demonstrate that existing applications can be “grid-enabled” with minimal changes, we have modified the GTK Display Interface for Structures (GDIS) to support

the launch of external computations on our mini-grid. GDIS is an open source program that allows scientists to visualise molecules, crystals and crystal defects in three dimensions and at the atomic scale [18]. Users can read output from a number of simulation codes including GULP, SIESTA and GAMES, in order to create the model. In addition to reading code output, GDIS can act as an interface to create code input and run the simulation on the local machine. In order to enable users to continue to interact with the graphical interface as the calculation is being run, the code is multithreaded. When an external task is launched, one of these threads forks a new process

that runs the task and blocks until the task is completed. An additional model representing the results of the calculation is then created in the graphical interface. These simulations can be computationally intensive and take several hours.

Clearly this approach makes it relatively easy to run a variety of simulations; however, the user is limited to running only a few calculations at once. In order to overcome this limitation we have modified GDIS to launch the calculations via our client tool. Modifications to the GDIS code base were limited to changing a single subroutine that launches each task from the queue of tasks to be run from the queue. Jobs are specified simply as a collection of arguments passed to our command line tools.

We require our client tool to upload data generated by GDIS, build the job description and submit it to the orchestration services and retrieve the output upon job completion so that it can be fed back into GDIS. The existing multithreaded machinery within GDIS allows many remote tasks to be simultaneously executed across the available grid resources. It is worth noting that the total time to modify the GDIS code base was only a few hours.

## 5. Conclusion

### 5.1 Related Work

An initial requirement analysis for the integrated use of *e*Minerals resources was defined and

presented in [2]. A solution, named `my_condor_submit`, was implemented as a submission wrapper over the Condor-G grid scheduling tool. It made use of Condor's workflow management tool (DAG-Man) and SRB client tools to schedule data transfers to and from the SRB data vaults alongside job executions.

This proved however to have several limitations: users were required to have a wide range of software available on their local machines, and required open two-way communication channels with all resources. We have aimed to solve this issue by relying on a BPEL engine to orchestrate resource invocations on behalf of the users. Further more, the combined use of DAG-Man and Globus proved to be relatively un-scalable in firewalled environments, with the number of simultaneous submissions limited by the number of open ports between client and resource. Finally, we introduce further refinements through the introduction of a wider range of services such as predictive metascheduling and the use of `myProxy` (which facilitates amongst other things the renewal of proxies).

A mention should be made of other attempts to develop lightweight grid client side tools. In particular the *Grid Resources On Workstation Library* (GROWL) package [19], which now includes the RCommands, aims to provide a commodity library against which grid enabled applications can be developed. Individual Web Services are used as middle tiers between users and existing grid middleware. While there are some basic architectural similarities, our workflow approach differentiates us by ensuring an integrated use of services, as opposed to providing a collection of semi-independent services and APIs.

## 5.2 Future Directions

Our work has been focused on demonstrating the potential use of the BPEL and Web Service standards to facilitate desktop access to grid infrastructures. We aim in future work to further refine many aspects of our current implementation. In particular, we will consolidate all security elements to rely solely on certificates as a common authentication mechanism. We also intend to target potential scalability issues by focusing on medium and large-scale processes. As a use case, we intend to address the following combinatorial problem: it has been suggested that screw dislocations in zeolitic materials may provide pathways to more rapid diffusion than through the undistorted

channel system with implications for there specify as ion exchange agents and gas separation membranes. We will test this hypothesis by studying the potential energy surface traversed by a CO<sub>2</sub> molecule moving through the perfect channel system and along the core of a screw dislocation.

Such problems are usually tackled by scientists by writing a simple script to generate the required input files and run calculations, either sequentially on the desktop machine or via a local queuing system. We intend to reproduce these using our client tools to schedule jobs remotely on our grid infrastructure.

## References

1. Dove, M. et al., Environment from the molecular level: an eScience testbed project. *Proc. of All Hands Meeting, Nottingham, 2003*
2. Blanshard, L. et al., Grid tool integration within the eMinerals project, in *Proc. Of the All Hands Meeting Nottingham, 2004*.
3. Andrews, T. et al., Business Process Execution Language for Web Services Version 1.1 OASIS, 2003.  
<http://ifr.sap.com/bpel4ws>.
4. The GridSAM project. <http://www.lesc.ic.ac.uk/gridsam/>
5. The Globus Project. <http://www.globus.org/>
6. Marmier, A., The eMinerals mini-grid and the National Grid Service: a user's perspective, in *Proc. Of the All Hands Meeting, Nottingham, 2005*.
7. White, T. et al., eScience methods for the combinatorial chemistry problem of adsorption of pollutant organic molecules on mineral surfaces, in *Proc. Of the All Hands Meeting, Nottingham, 2005*.
8. Murray-Rust, P. and Rzepa, H., Chemical Markup Language and XML Part I. Basic principles, *J. Chem. Inf. Comp. Sci.*, **39**, 928, 1999.
9. Soler, J. et al., The SIESTA method for ab initio order-N materials simulation, *J. Phys.: Condens. Matter* **14**, 2745 (2002).
10. Gale, J. et al., GULP – a computer program for the symmetry adapted simulation of solids, *JCS Faraday Trans.* **93**, 629, 1997.
11. Chapman, C. et al., Condor Birdbath - web service interface to Condor, in *Proc. of the All Hands Meeting, Nottingham, 2005*.
12. Emmerich, W. et al., Grid Service Orchestration using the Business Process Execution Language (BPEL), *UCL-CS. Research Note RN/05/07*, 2005.
13. Anjomshoaa, A., et al., Job Submission Description Language (JSDL) Specification v1.0, GFD #: GFD-R-P.056, 2005. [http://www.ggf.org/ggf\\_docs\\_final.htm](http://www.ggf.org/ggf_docs_final.htm)
14. Couch, P., et al., Towards Data Integration for Computational Chemistry, in *Proc. Of the All Hands Meeting, Nottingham, 2005*.
15. Java Cog Kit <http://wiki.cogkit.org/>
16. Apache Software Foundation. <http://www.apache.org/>
17. MyProxy <http://grid.ncsa.uiuc.edu/myproxy/>
18. Fleming, S., and Rohl, A., GDIS: a visualization program for molecular and periodic systems, *Z. Krist.* **200 580**, vol.220 pp.580-584, 2005.
19. Hayes, M. and Morris, L., GROWL: A Lightweight Grid Services Toolkit and Applications in *Proc. Of the All Hands Meeting, Nottingham, 2005*.