IWPLS'09

Pages 1–8

# Rapid development of computational science portals

Jos Koetsier and Jano I. van Hemert*

University of Edinburgh, School of Informatics, Informatics Forum, 10 Crichton Street, Edinburgh EH8 9AB, United Kingdom

September, 2009

Published in the IWPLS'09 International Workshop on Portals for Life Sciences

**ABSTRACT**

**Motivation:** Scientific web portals are seen as the way forward to improve upon the slow uptake in use of utility computing infrastructure and high-performance computing facilities (Wilkins-Diehr, 2007). Currently, two types of portals exist: general-purpose portals and domain-specific portals. The first type closely resembles the underlying technical infrastructure of compute-job submission systems, thereby providing little appeal to a wide range of domain specialists. The second type is tailored to the application specifications and their end-users' requirements. Unfortunately, the technical complexity in domain-specific portals makes these expensive and time-consuming to develop and maintain. Clearly, an alternative to these two approaches is required.

**Results:** We introduce an approach, Rapid, that facilitates rapid development of portlets. Its main aim is to reduce the time from development to the deployment from several months to a few weeks. Moreover, it facilitates an easy way to share and maintain these portlets by domain specialist themselves. Both these advantages considerably reduce the cost of developing portal solutions for computational science applications. We highlight several scientific domains where our approach is used or was used successfully.

**Availability:** Rapid is developed under an Open Source model and is available freely through a Gnu General Public license. Main releases, documentation, tutorials and examples are available at research.nesc.ac.uk/rapid. The development of Rapid uses an open read-only CVS repository, which is complemented by a developer community site at forge.nesc.ac.uk.

**Contact:** j.vanhemert@ed.ac.uk, jkoetsie@ed.ac.uk,

## 1 INTRODUCTION

Recently, web portals have gained significantly in popularity. Currently, the largest companies in the web search business deliver their content through portal technology, including Google with their iGoogle portal, Microsoft with MSN and Yahoo. Portal technology has also gained popularity in social networking sites such as LinkedIn and Myspace and other sites such as the BBC and many more.

A web portal is a framework that combines the output of several independent web applications and arranges them in one or more customisable web pages. The output of each web application tends to be rendered surrounded by a border that gives it the appearance of residing in a window. Depending on the portal technology used, these web applications are referred to as 'portlets', 'widgets' or sometimes 'gadgets'. We will use the term 'portlet' throughout this paper.

Usually portal frameworks offer users the possibility to log in to a personalised environment using a username and password. When logged in, users can customise their own pages by selecting the layout and deciding which portlets to display and where. The more advanced portals can do this using user friendly menus and drag-and-drop.

There are a number of standardisation efforts underway to enable portlets to be used in portals from different vendors, using the principle of 'write once, deploy anywhere'. Currently there are the JSR 168 (Abdelnur and Hepper, 2003) standard and its successor JSR 268 (Nicklous and Hepper, 2008), WSRP (OASIS, 2008) and the OpenSocial (Google, 2009) standard. Several open and closed source products exist that implement these standards and which can use conforming portlets. Examples of such frameworks include Liferay, GridSphere, JBoss and WebSphere.

The scientific community have long since recognised the advantages and potential of using portal technology in scientific applications such as described in Wege (2002), especially as a tool to improve upon the slow uptake in use of utility computing infrastructure and high-performance computing (HPC) facilities (Thomas *et al.*, 2005). Instead of learning command-line methods to upload and execute applications, users would have one web-based point of access to the available computational resources. Users can select the applications that are of interest to them and use a graphical user interface to execute a particular task, substantially lowering the barrier to using the available computing resources and enhance scientific research.

Unfortunately writing new scientific portlets is not an easy task. They are usually written in Java or JavaScript and, for more advanced portlets, require the use of several middleware packages. Once written, they need to be maintained as depending technologies may be updated or changed. An easy solution is to program a few portlets that are very general in their application and cover a wide range of possible end-uses. Unfortunately, this means they necessarily closely resemble the underlying technical infrastructure of compute-job submission systems, thereby providing little appeal to a wide range of domain specialists.

In this paper we present a solution to this problem by introducing a novel methodology, 'Rapid'. Its main aim is to enable cost-effective and rapid development of job submission portlets that are tailored to enable domain specialists to accomplish their domain-specific task. In other words, users of job submission portlets

*to whom correspondence should be addressed

created with Rapid should not need to understand any underlying technology. Its secondary aim is to make this method of delivering scientific web portals simple enough to allow adoption by domain experts with little background in ICT. This will enable communities to develop and maintain their web portals independently.

In this paper we will refer to people involved in the study by roles. Developers are those who are involved in the design and development of the Rapid method. Domain-specialists, researchers and teachers are the Portal Users, e.g. computational chemists, brain imaging experts and biologists. These will work with the portlets created with Rapid. A Portal Designer uses Rapid to design, develop and deploy portlets for Portal Users. When the Portal Designer in a project is also a domain-specialist we have accomplished our secondary aim.

We demonstrate how Rapid is used to develop and deploy portlets for computational science portals. We highlight a number of use cases in several scientific domains where Rapid is used or was used successfully to deliver a solution.

## 2 PORTAL DEVELOPMENT WITH RAPID

In Figure 1 we illustrate the processes of creating, deploying and using a domain-specific portlet using the Rapid system. It is important for understanding the philosophy behind Rapid to show a typical way a portlet will be used by a Portal User. The focus of the methodology is to create portlets that facilitate customised computational tasks, where they are completely shielded from the infrastructure. It is up to the Portal Designer to decide how much to shield the Portal Users from the underlying applications. For instance, a Portal Designer may decide to expose a very limited subset of an application's feature to ensure inexperienced users can perform a number of specified tasks.

The process of creating and using a Rapid portlet entails a number of steps. In the design phase the Portal Designer specifies (1) the user interface and logic of a task. This specification is created using *Extensible Markup Language* (XML). The Portal Designer executes (2) the Rapid Translator, which reads (3) this specification, creates the portlet and generates a Java WAR file, which is then deployed (4) into a portlet container. The generated portlet uses the JSR 168 standard, which ensures it can be deployed in a variety of different portal containers. However, as the packaging of the WAR file and the requirement of different deployment descriptors for each portal, separate packaging options are available to generate specific WAR files.

Once deployed, a Portal User can use a web browser to log in to the portal and access (5) the new portlet. Using domain-specific jargon and graphical user-interface elements such as drop-down menus, radio buttons and check boxes, (s)he configures and then performs (6) the task at hand. This transfers control to the computational task manager embedded in the portlet, which runs (7) the appropriate compute jobs on available compute resources. The portlet allows monitoring (8) of the progress of all tasks submitted so far, and when the compute jobs finish the results can be transferred (9) to an appropriate location and analysed (10) by embedding web-based visualisation components.

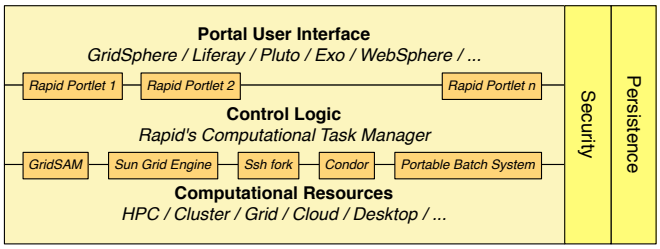


**Fig. 2.** The conceptual layers involved in Rapid portlets; Rapid portlets provide the link from the user interface, i.e. the portal, to the computational resources, e.g. HPC, Cluster, Grid, Cloud or Desktop resources. It does so via communication with the resource managers, e.g., GridSAM, Sun Grid Engine, Condor, Portable Batch System and direct forks over Ssh.

## 3 ARCHITECTURE OF RAPID

In Figure 2, we depict the typical design of computational science portals. The top layer provides the actual user interface, which is delivered via a JSR 168 compliant portal framework. Many frameworks are freely available, some with the possibility to purchase commercial support, such as the JBoss portal, Liferay and Gridsphere. Commercial offerings include IBM's WebSphere and Oracle's Portal server. Rapid generates portlets that live inside any of these frameworks.

The control logic layer orchestrates the actual business logic of the portlet and consequently is responsible for data management, configuring and submitting the computational job. Usually, submitting a computational job entails copying data and applications into the computational resource, initiating and monitoring execution and copying results to a resource where they can be visualised as is specified in Foster *et al.* (2007).

Security spans all layers as a user that authenticates in a portal eventually must also successfully authenticate with all the resources required to perform a task. This means that credentials may have to be taken from the interface layer all the way down to the resources level. However, in some situations specific authentication may be required if portal credentials are not recognised by certain resources. Rapid allows for this by providing variables together with login and password input boxes that can be made available within portlets.

Persistence too must span all layers as Portal Users will undoubtedly expect their previous computational tasks, and more importantly the results from these tasks, to be available next time they login to their science portals. To do so, the persistence in Rapid keeps records of all that happens in portlets and is able to communicate directly with the resource layers to update these records when computer jobs progress.

Using our method, the Portal Developer defines the user interface in an XML configuration file and the Portal user fills in the web forms resulting from this definition. The business logic takes the contents of these forms as parameters and configures, submits and monitors computational jobs. The final layer in our model consists of the file and compute resources that Rapid uses to perform its tasks. An UML component diagram with these components is shown in Figure 3

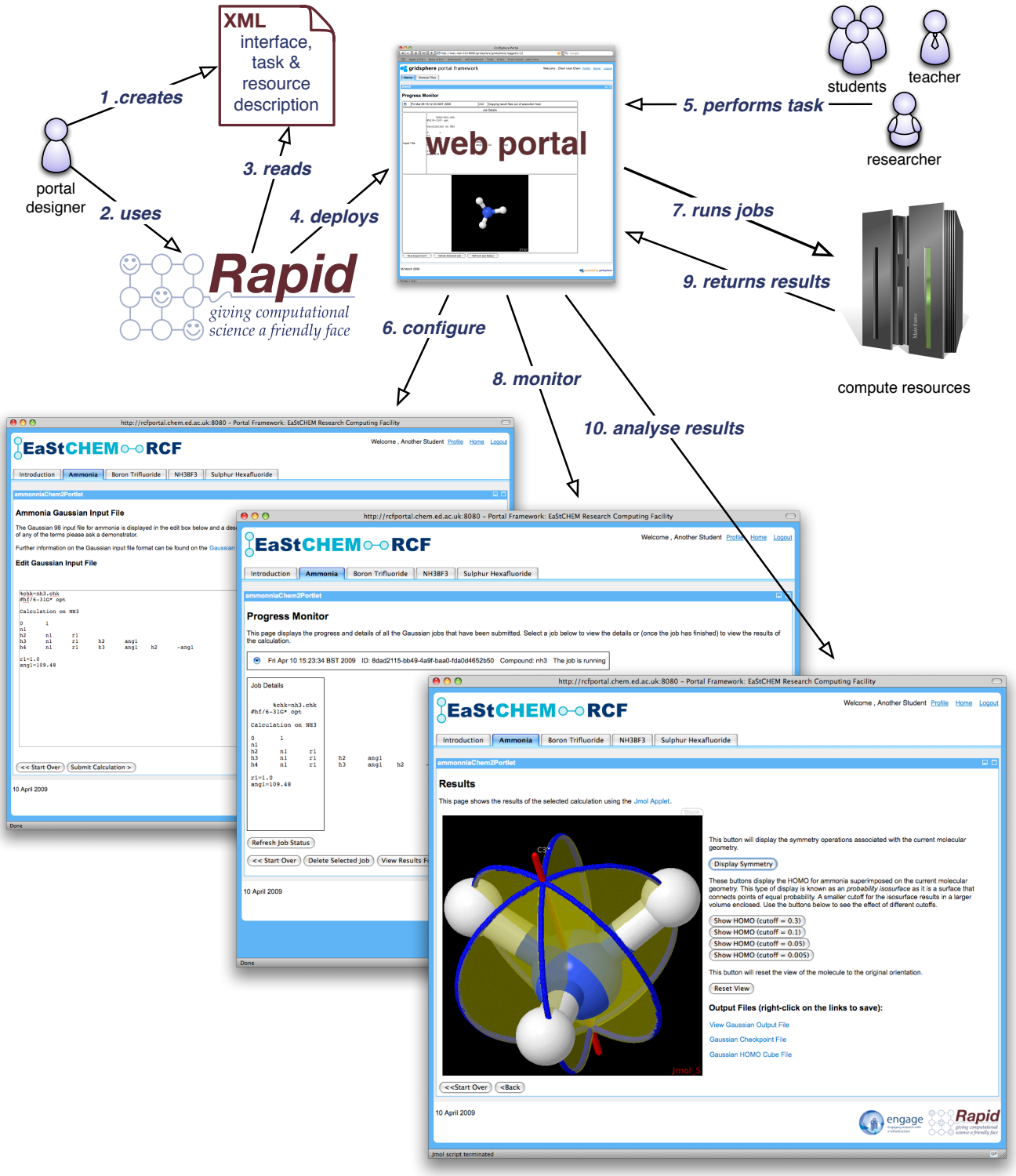


**Fig. 1.** The process of developing, deploying and using portals with Rapid. Two main roles are identified: the Portal Designer and the Portal User, shown here as students, teachers and researchers

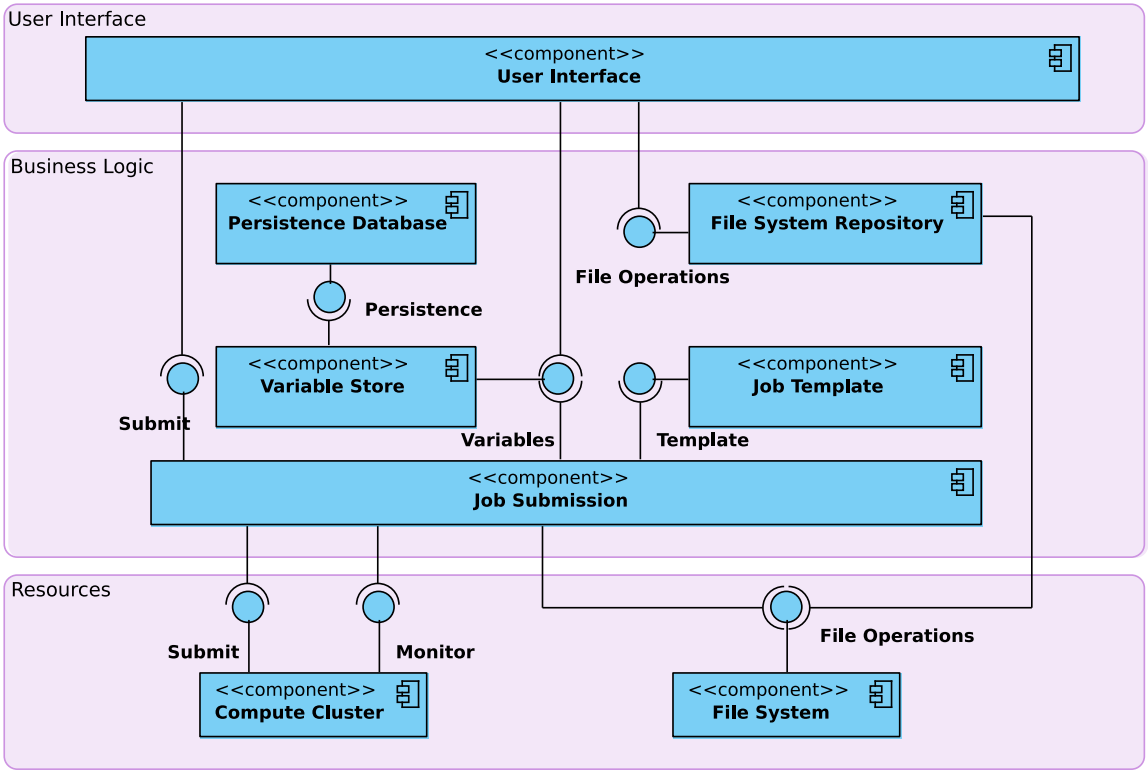


**Fig. 3.** UML Component diagram of Rapid, which shows how specific components fit into the user interface, business logic and resources layers

### 3.1 User interface layer

The user interface is specified in the Rapid XML file. Rapid separates the mark-up of the user interface and the logic that communicates with the underlying business logic.

The mark-up of the user interface is specified using tags from the standard XHTML namespace. This gives a Portal Designer a wide range of options to design the user interface using familiar tags. For example, a Portal Designer with basic XHTML knowledge can easily create tables, use different fonts and incorporate images in the Rapid Portlet without the need to learn a new computer language.

Tags from the Rapid namespace are used to enter new values and to communicate with the job submission engine. Parameters can be requested in different ways, such as text boxes, list structures and radio buttons. Default values can be set in advance and are retained during the session. As HTTP is a stateless protocol this would normally have to be programmed explicitly by tying each variable into the session set up by the portal. Values entered can be checked whether they are valid by comparing them to a regular expression. This ensures that users cannot, for example, enter text where a number is expected.

Using Rapid tags, the portlet can be subdivided into a set of "pages". This allows the Portal Designer to guide the user through the process of submitting a task. Instead of displaying all available information at once, a small amount of information can be requested of the user at one time. When all required information has been supplied, the user clicks on a button and navigates to a new page, which in turn can ask for more information, request confirmation or display monitoring information.

### 3.2 Business logic layer

Rapid's business logic decides what to do with the values entered in the User Interface. As the main purpose of Rapid is job submission, a very general model of a job submission is used. The model needs to be flexible enough to allow a wide range of possible scenarios, but on the other hand specific enough to ensure creating a new job submission portlet does not lose any simplicity,

To achieve both goals, Rapid's job submission architecture is based around two Open Grid Forum standards: the *Basic Execution Engine* (BES) (Foster *et al.*, 2007) and *Job Submission Description Language* (JDSL) (Anjomshoaa *et al.*, 2005). These standards, which describe a general framework for submitting compute jobs, were carefully designed based on years of experience with a variety of different kinds of computational resources. This ensures our model can cope with most computational scenarios and underlying computational infrastructures. Together these standards define a procedure of job submission that involves the following stages:

*Stage In,* a number of files are transferred into a computational resource that is expected to execute the compute jobs that belong to the task.

*Execute,* the computational resource is executing the task.

*Stage Out,* the task has finished and the results of the task are transferred to another site.

*Error,* this state is entered whenever an error has occurred in one of the previous states.

The Portal Developer specifies a job template in the Rapid XML file along the lines of the JSDL specification. This includes the name and location of the application itself and the file transfers before and after the execution. The application itself is described as a POSIX job such as used in most UNIX systems: the name of an executable, a set of parameters and environment variables.

### 3.3 Resource layer

If so desired, the compute job can be performed on the same computer the actual portal is running on, but in the majority of the cases the compute job is to be offloaded to a dedicated resource. This can either be a single computer accessed through SSH or a compute cluster running a workload management system that distributes and monitors the jobs.

Rapid has an internal engine that can access several high performance and high throughput cluster technologies. These include Condor, Sun Grid Engine and the Portable Batch System (PBS). All of these manage compute jobs on local clusters. The computational task manager of Rapid communicates directly with all these solutions, initiates the data transfers and polls the resources to determine which state the execution is in. It is possible to define several computational resources in the Rapid XML file and let the Portal User choose between them.

On the other hand, GridSAM can be used to bypass most of Rapid's internal job manager. GridSAM is an implementation of the Basic Execution Engine and provides a Web Service interface that consumes JSDL XML documents and performs data staging and job submission to several Grid Computing infrastructures. When submitting to a GridSAM web service, Rapid creates a JSDL document and hands it to a GridSAM service, which will perform the actual data staging, job submission and monitoring.

Transferring files to and from the computational resources can be performed using standard protocols such as FTP, SFTP, HTTP and the Grid based GSIFTP. If the computation takes place on the portal itself, simple operating system level 'copy' commands can be used.

### 3.4 Example

We present an abstract example of how a Portal Developer would create a portlet. In the XML file we configure a job which simply echos a string. In the job template we specify an executable '/bin/echo' with parameters 'hello' and a reference to a value (See Figure 4(a)). The variable this example refers to has to be initialised and given a reasonable default value, which in this case will be set to 'world' (See Figure 4(b)).

Finally, we allow the user to change the second parameters using a drop down box, with choices 'World', 'Rapid' and 'Unknown' (See Figure 4(c)). The framework ensures the right default value is presented to the user and that any changes the user makes, will be retained throughout the configuration of the job.

This example shows how we can mix the XHTML namespace and the Rapid namespace to separate mark-up from Rapid's logic. The first line is using the well known `<H1>` element to enlarge the sentence 'Enter missing word'.

```
<posix>
    <executable>
        <single><value>/bin/echo</single><value>
    <parameter index="0">
        <single><value>hello</value></single>
    </parameter>
    <parameter index="1">
        <single><value>$(variable)</value></single>
    </parameter>
</posix>
```

(a) Specifying the executable and parameters in Rapid. The second parameter refers to a variable.

```
<initialise>
    <variable name="variable">
        <single><value>World</value></single>
    </variable>
</initialise>
```

(b) Initialising a variable.

```
<x:h1>Enter missing word</x:h1>
<variable name="variable">
    <list size="1">
        <item value="World">World</item>
        <item value="Rapid">Rapid</item>
        <item value="Unknown">Do not know</item>
    </list>
</variable>
```

(c) Adding a drop-down box. The target is variable with name 'variable'. The `<x:h1>` element belongs to the XHTML namespace.

**Fig. 4.** Examples of Rapid XML

```
from uk.ac.nesc.rapid.plugin import RapidPlugin
import datetime
class SetTime(RapidPlugin):
   def doAction(self):
       today =  datetime.date.today()
       self.addHTML("<H1>Today's date is")
       self.addHTML(today)
       self.addHTML("</H1>")
```

**Fig. 5.** Simple example of Jython code, which enables inserting the current date dynamically into a page

### 3.5 Plug-in functionality through Jython

The Rapid methodology gains most of its power by encapsulating common programming constructs and using XML tags to invoke them. Each XML tag represents a substantial amount of Java code, which means that job submission portlets can be built very quickly. However this also results in a loss of flexibility: programming constructs that have not been implemented through XML tags cannot be used. This includes the use of looping mechanisms, arithmetic and functions such as database access.

Rather than anticipating and implementing every possible need and thereby implementing a complete new computing language,

a plug-in framework was developed based on Jython (a Python implementation in Java). Within a plugin, a Portal Developer can query Rapid's Job Submission and Monitoring component and read and write to all variables. Of course the programmer can also use many of the existing Python and Java libraries.

Plug-ins can manipulate the user interface, not only adding output in the form of HTML but also by adding and removing input elements such as text input boxes and radio buttons. This feature allows run-time building of the user interface enabling the user interface to look differently, depending on certain conditions. An example of Jython is shown in Figure 5, where the current date is inserted dynamically in the page. To make use of this Jython code, the Portal Designer simply inserts `<plugin file="date" class="getDate"/>` at the appropriate location in the Rapid-XML.

## 4 USE CASES

Rapid has been used to build scientific portals in several specialist domains. We highlight five of these below where we give a general description of the tasks these portals enable and provide an approximation of the duration of the development period. Each portal was created by one separate Portal Designer; the names of these and all people involved are in the acknowledgements.

Chemists from the joint Chemistry Research School of the Universities of Edinburgh and St. Andrews (EaStCHEM) have recently used Rapid to create portlets both for teaching and research purposes. Technological barriers prevented many researchers from using specific chemistry applications and produced large overheads in teaching these applications to students. To overcome these barriers, the Rapid project showed EaStCHEM how to create portlets with Rapid. This training led to a technology transfer, where the most ICT adapt chemist was able to create several portlets without the help of a software developer within a period of two weeks. One of the portlets is now used to teach over 140 chemistry students. This user-friendly portlet, which is shown in the bottom half of Figure 1 hides the complexity of the tutorial (such as, command line options, authentication protocols and job submission commands) from the student, so that they can concentrate on learning both the conceptual use of the application and the chemistry associated with it. A screencast is available at research.nesc.ac.uk/node/396. Portlets do not just make applications more user friendly. Future work with EaStCHEM will make more resources available to the chemists, such as those supplied by the National Grid Service. This will overcome difficulties in securing time on the resources available within the universities and will harness more computational power for the chemists.

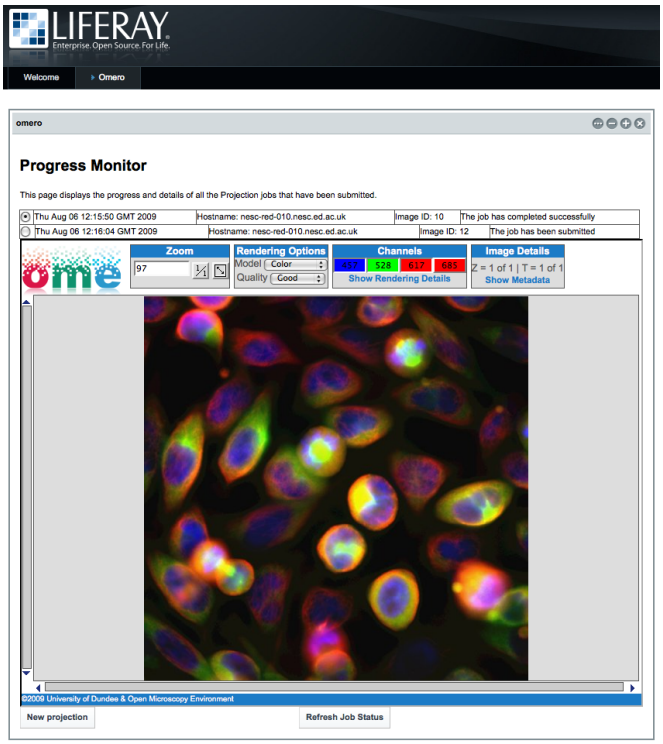


**Fig. 6.** Screenshot of the computational resource selection part of the portal created by Rapid for image processing in microscopy

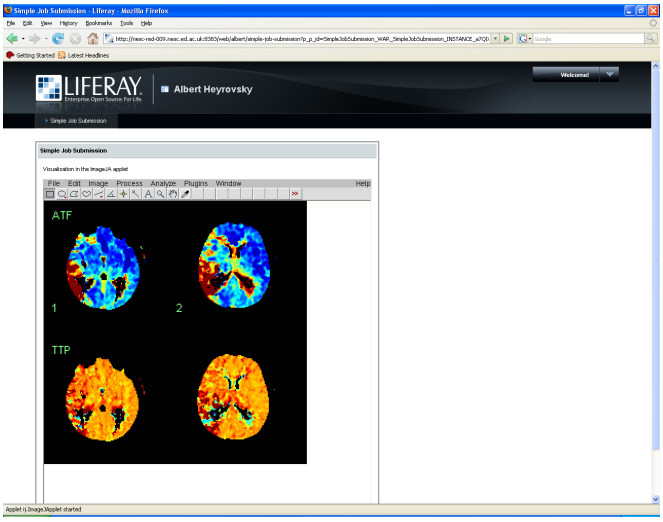


**Fig. 7.** Screenshot of the visualisation part of the portal created by Rapid for perfusion imaging in stroke treatment

In the context of biology, Rapid was used to create a portal for microscopy. Here the goal was to improve the usability of computationally-intensive image processing techniques. These techniques were developed by the Swedlow Lab at the University of Dundee under the Open Microscopy Environment (openmicroscopy.org). Using Rapid, a portlet was developed that integrates the analysis and visualisation techniques into one task that is executed on their local cluster. In Figure 6, we show the step where Portal Users can select which computational resources to use. The development of this portal took three weeks.

Rapid is being used to design, build and deliver a portal to help brain imaging experts perform tasks using imaging tools developed by the SFC Brain Imaging Research Centre (www.sbirc.ed.ac.uk) at the University of Edinburgh. These tasks are concerned with CT and MRI brain scans in the context of stroke; they include

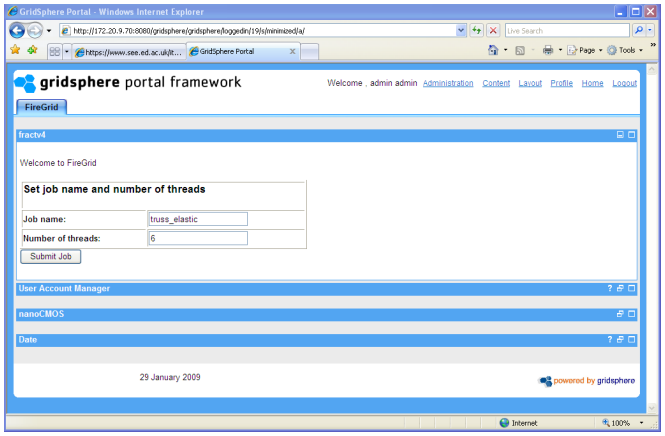


**Fig. 8.** Screenshot of a parameter setting part of the portal created by Rapid for running simulation in fire safety engineering

volume registering, image registration, region of interest selection, quantification of perfusion and the visualisation of the results. A screenshot of this portal is shown in Figure 7. It was created by an e-Science MSc student again in a period of four weeks.

Rapid is also being used in the FireGrid (firegrid.org) project (Wickler *et al.*, 2009) at the University of Edinburgh. It aims to establish a cross-disciplinary collaborative community to pursue fundamental research for developing real time emergency response systems, using the Grid, beginning with fire emergencies. Rapid was used to create a portal for running nonlinear finite element analysis using Abaqus, which took five weeks to develop and deploy. Figure 8 shows the parameter setting step in the task plan.

The Rapid Portals for Seismological Waveform Data project (research.nesc.ac.uk) aims to encourage the seismology community to use an application that allows analyses of seismic waveform data. This will be achieved by embedding the analysis application in a community gateway, which already exists in the form of a web portal. The main barriers to uptake here are the installation and understanding of the analysis package, the difficulty in transporting large amounts of data as input to an analysis, and direct visualisation of the results. Self-contained web portlets will be directly embedded in the community gateway and link to the data available in the Orfeus Data Center (orfeus-eu.org)—the primary European centre for this kind of data. All the data and computing will be orchestrated via Rapid, and the results presented to seismologists via their existing web portal.

## 5 DISCUSSION

Scientific computing is becoming a prevalent methodology in many disciplines (Getov, 2008). Today there is a wide range of new scientific areas, including Bioinformatics, Systems Biology, Computational Chemistry, Biomedical Imaging, Earth Systems Modelling, Computational Neuroscience, and Computational Physics. This has led to a surge in demand for computational infrastructure, which in turn has resulted in a growth in computational resources, such as those offered through utility computing and high performance computing facilities. Although new computing infrastructure is rapidly being deployed, its broad uptake is not following at the same pace (Simpson *et al.*, 2008). The main reasons for this lie in the complexity of the human interface to these computing infrastructures and the predominant focus on developing the underlying fabric (see for example Matsuoka *et al.* (2008)) as opposed to user interfaces.

Most computational resources rely on a command-line interface or an application programming interface as the primary route to operation. As scientists from various disciplines are not always familiar with such types of interfaces, the current approach is to operate via a third person. This is a slow and expensive route, which is often limited by the length of a specific research project.

Another approach, which has recently gained popularity, is to offer access to computing infrastructures via portals (Novotny *et al.*, 2004). A portal is essentially a web-based application that acts as a container for page fragments generated by a set of Java-based components called portlets. Portals usually offer a rich set of features such as user management, security and a set of personalisation features that allow a user to customise which portlets to render and how to display them. Through the use of a portal, we can offer a community of scientists easy, secure web-based access to underlying computing infrastructure.

We introduce an approach that allows rapid development and deployment of portals for scientific computation. It uses a philosophy different from any other portal delivery system currently in existence. Whereas most portals are graphical representations of the underlying job submission components, our system delivers portals that allow domain specialists to perform tasks in the context of their own domain. The benefits are: shorter development to deployment periods, easier maintenance and a unified approach to application usage.

## 6 CONCLUSIONS

We have introduced an approach that facilitates rapid development of portlets, where the time to deployment can be reduced from several months to a few weeks. Moreover, as the whole portlet is declared in one file, it is easy to share and maintain these portlets. Both these advantages considerably reduce the cost of developing portal solutions for computational science applications.

Our technological embodiment of our approach is Rapid, which was used to create computational science portals in five different specialist domains. It has delivered portlets for all these portals without any conventional programming required.

These portal allow scaling the uptake of software applications in these communities much faster than when they would have to rely on a small number of expert users to drive the software. In other words, making the applications easier to use by wrapping them in standard tasks is more cost-effective than training everyone to use the applications.

By successfully transferring our technology to the domain specialists in computational chemistry we have enabled a second scaling effect. This community can now increase the number of tasks enabled through web portals by themselves. This frees up the Rapid team's time to develop Rapid and apply it to other communities. We aim to make these technology transfers happen in these communities too.

## ACKNOWLEDGEMENTS

### Collaborators

We thank the following collaborators for their effort in successfully getting the use cases to work. Portal Designers are shown in *italics*.

Chemistry: *Andrew Turner*, Patricia Richardson, Neil Robertson and Carol Morrison (University of Edinburgh); Sharon Ashbrook (St. Andrews).

Brain Imaging: *Albert Heyrovsky*, Trevor Carpenter, David Rodríquez González, Joanna Wardlaw (University of Edinburgh).

Microscopy: *Donald MacDonald* and Jason Swedlow (University of Dundee).

FireGrid: *Bruce Duncan* and Asif Usmani (University of Edinburgh).

Seismology: *Alessandro Spinuso* and Torild van Eck (Observatories and Research Facilities for European Seismology); Andy Heath (University of Liverpool);

### Funding

The development of Rapid was funded initially as a Commissioned Software Project in the OMII-UK Centre and Managed Programme (EPSRC, EP/D076617/1). Further development and deployment of Rapid was funded as an ENGAGE Project under the Engaging research with e-Infrastructure project of the Joint Information Systems Committee (JISC) and as a Rapid Innovation project under the JISC Information Environment Programme. It is also supported by a Research Platform Grant (EPSRC, EP/F057695/1) under the e-Science Core Programme.

We acknowledge the sponsorship of The e-Science Institute and the Scottish Bioinformatics Forum to the 2009 International Workshop on Portals for Life Sciences.

## REFERENCES

Abdelnur, A. and Hepper, S. (2003). JSR 168: Portlet specification. http://www.jcp.org/en/jsr/detail?id=168.

Anjomshoaa, A., Brisard, F., Drescher, M., Fellows, D., Ly, A., and McGough, S. (2005). Job Submission Description Language (JSDL) Specification, Version 1.0. Technical report, Global Grid Forum. http://www.ogf.org/documents/GFD.56.pdf.

Foster, I., Grimshaw, A., Lane, P., Lee, W., Morgan, M., Newhouse, S., Pickles, S., Pulsipher, D., Smith, C., and Theimer, M. (2007). OGSA Basic Execution Service Version 1.0. https://forge.gridforum.org/sf/go/doc15062?nav=1.

Getov, V. (2008). e-science: The added value for modern discovery. *Computer*, **41**(11), 30–31.

Google (2009). OpenSocial v0.9. http://www.opensocial.org/.

Matsuoka, S., Saga, K., and Aoyagi, M. (2008). Coupled-simulation e-science support in the NAREGI grid. *Computer*, **41**(11), 42–49.

Nicklous, M. and Hepper, S. (2008). JSR 286: Portlet specification 2.0. http://www.jcp.org/en/jsr/detail?id=286.

Novotny, J., Russell, M., and Wehrens, O. (2004). Gridsphere: a portal framework for building collaborations: Research articles. *Concurrency and Compututation: Practice & Experience*, **16**(5), 503–513. System available at http://www.gridsphere.org/.

OASIS (2008). Web services for remote portlets specification v2.0. http://docs.oasis-open.org/wsrp/v2/wsrp-2.0-spec.html.

Simpson, A. D., Bull, M., and Hill, J. (2008). Identification and categorisation of applications and initial benchmarks suite. Technical report, EPCC, University of Edinburgh, UK.

Thomas, M. P., Burruss, J., Cinquini, L., Fox, G., Gannon, D., Gilbert, L., von Laszewski, G., Jackson, K., Middleton, D., Moore, R., Pierce, M., Plale, B., Rajasekar, A., Regno, R., Roberts, E., Schissel, D., Seth, A., and Schroeder, W. (2005). Grid portal architectures for scientific applications. *Journal of Physics: Conference Series*, **16**, 596–600.

Wege, C. (2002). Portal server technology. *IEEE Internet Computing*, **6**(3), 73–77.

Wickler, G., Beckett, G., Han, L., Koo, S. H., Potter, S., Pringle, G., and Tate, A. (2009). Using simulation for decision support: Lessons learned from FireGrid. In J. Landgren, U. Nulden, and B. Van de Walle, editors, *Proceedings of the 6th International Conference on Information Systems for Crisis Response and Management*.

Wilkins-Diehr, N. (2007). Special issue: Science gateways—common community interfaces to grid resources: Editorials. *Concurr. Comput. : Pract. Exper.*, **19**(6), 743–749.