Marek Kasztelnik*, Marian Bubak**,*

GRID RESOURCE REGISTRY – ABSTRACT LAYER FOR COMPUTATIONAL RESOURCES

The growing number of resources available to researchers in the e-Science domain has opened new possibilities for constructing complex scientific applications while at the same time introducing new requirements for tools which assist developers in creating such applications. This paper discusses the problems of rapid application development, the use of distributed resources and a uniform approach to resource registration, discovery and access. It presents the Grid Resource Registry, which delivers an abstract layer for computational resources. The Registry is a central place where developers may search for available services and from which the execution engine receives technical specifications of services. The Registry is used throughout the lifetime of the e-science application, starting with application design, through implementation to execution.

Keywords: *e-science, collaborative applications, distributed applications, resource discovery, registry, common information space*

GRID RESOURCE REGISTRY – ZUNIFIKOWANY DOSTĘP DO ZASOBÓW OBLICZENIOWYCH

Rosnąca liczba zasobów dostępnych dla naukowca z jednej strony otworzyła nowe możliwości w konstruowaniu złożonych aplikacji naukowych, a z drugiej przyniosła dodatkowe wymagania dla narzędzi wspierających proces tworzenia oraz uruchamiania takich aplikacji. W artykule przedstawiono wyzwania związane z szybkim wytwarzaniem aplikacji naukowych, które wykorzystują rozproszone zasoby oraz związane z nimi trudności wynikające z rejestrowania, wyszukiwania i wywoływania zasobów używanych przez aplikację. Rozważania przedstawiono na przykładzie Grid Resource Registry – centralnego rejestru, który dostarcza abstrakcyjnego opisu rozproszonych zasobów, dzięki czemu w znaczący sposób proces wytwarzania oraz uruchamiania aplikacji naukowych może zostać uproszczony.

Słowa kluczowe: *e-science, kolaboratywne tworzenie aplikacji, aplikacje rozproszone, odnajdywanie zasobów, rejestr, wspólna przestrzeń danych*

* AGH University of Science and Technology, ACC Cyfronet AGH, ul. Nawojki 11, 30-950 Krakow, Poland, m.kasztelnik@cyfronet.pl

** AGH University of Science and Technology, Faculty of Electrical Engineering, Automatics, IT and Electronics, Department of Computer Science, al. A. Mickiewicza 30, 30-059 Krakow, Poland, bubak@agh.edu.pl

1. Introduction

The nature and needs of e-science necessitate rapid development of applications which addresses specific problems. To fulfill this requirement, applications are built from preexisting computational blocks, such as libraries or distributed resources available online [7][9]. A single application may comprise various technologies implemented in many frameworks. Each of these technologies has different characteristics (such as statefulness or lack of it). Thus, the developer should always choose the most suitable technology for implementing a particular service.

The process of creating applications which use distributed computational resources is complicated by several issues. The first issue concerns communication between the end user and the application developer. In most cases there are many differences concerning the way in which the user understands application functionality and the way it is implemented by the developer. Another problem affects libraries and computational resources used in application development. There is no central repository which can be searched (in a user-friendly way) for computational blocks that fulfill developer needs, along with descriptions and additional functional (e.g. availability) or nonfunctional (e.g license, SLA) specifications. Furthermore, the nature of the distributed environment where resources are installed introduces additional complexity related to network, software and hardware availability. Last but not least, there is the issue of technical complexity of applications which reuse existing computational resources. As the number of available technologies grows the developer has to spend a significant amount of time mastering new technologies instead of implementing user needs. Furthermore, using different frameworks in a single application often leads to incompatibilities between libraries and configuration issues.

In light of the problems presented above, there is a need for a module which would enable the application developer to query for and manage information about distributed computational resources. In this paper we present a customized registry called the Grid Resource Registry (GRR) and focus on cooperation between different groups of users during application development, discovery, registration and execution on distributed computational resources. The key scientific objective of our work focuses on ensuring simplicity of e-science application development and on a novel approach to uniform resource description and access.

The next section presents an overview of existing solutions for resource discovery and for simplifying the process of application development. Sections 3 and 4 describe requirements applicable to our registry in the scope of the ViroLab [5] and Gredia [16] projects as well as the abstraction used in this solution. The proposed architecture of our system (implementing the presented assumptions) is described in section 5. We also present an application development case study (section 6). Finally, section 7 presents a summary and a discussion of planned improvements to the repository.

2. Related work

The issue of rapid application development is intimately tied with the Software as a Service (SaaS) paradigm, which has been gaining popularity over the recent years. It should therefore come as no surprise that many interesting ideas and systems have already been proposed in this context.

Searching for information about distributed computational resources is one of the most time-consuming steps in the development process. The Universal Description, Discovery and Integration registry (UDDI registry) [19] addresses this need by enabling users to store and query business information about service providers. The standard has many implementations delivered by various companies such as IBM, BEA, HP, Oracle and Microsoft. In addition to commercial products, there are also many open-source implementations, including Apache jUDDI [8] and Grimoires [18]. UDDI should therefore be considered a popular standard for storing information about services. Unfortunately, most UDDI installations are internal to commercial organizations (and usually hold only information on the organization's resources) – thus, it is not possible to search for and reuse such information outside a given organizational unit. Another relative disadvantage is the lack of semantic information about service input and output. As a result, the developer has to search for such information manually, e.g. by browsing service WSDL files. Moreover, this solution only supports plain web services.

Feta [13] goes further by enabling users to search for information using complex queries, with constraints applied to input and output parameters as well as to service behavior. Owing to integration with the Taverna workflow composition system [17] it is a powerful environment, facilitating fast workflow development. It is not, however, devoid of drawbacks as Taverna only supports web services. Another issue related to Feta is its preferred methodology of application development – namely, workflow composition. This approach enables users with limited computer skills to compose simple applications, but it also introduces problems related to managing complex workflows with `if` and `loop` conditions, and data type conversions.

The process of application development can be simplified even further by delivering an environment capable of composing applications in an automatic way. The user only defines input parameters and the expected result format while the system locates computational resources which can be used to produce the requested result. Examples of such systems are proposed in [4] and [12].

The challenge of storing descriptions of computational resources in one place is also addressed by BioCatalogue [3], EMBRACE [6], SeekDa [20] and Distributed Grid Resource Registry Meta-Service [10] projects. These registries focus on registering and querying information about web services. SeekDa aims to publish a one-stop service marketplace, enabling service providers to sell computational resources.

The presented solutions, despite many advantages, do not fulfill all the requirements of e-science research teams: specifically, they do not address support for collaborative application development and access to different technologies in a uniform

way. Furthermore, they only allow users to search for specific installations of a given service. There is no abstract layer of distributed computational resource description to dynamically locate the best (and working) service instance in the application execution phase.

3. Identification of challenges

Taking into account the advantages and drawbacks observed in existing solutions we have decided to design a new registry which helps develop collaborative applications. The premise of our system is to allow researchers to define requirements for the application in an easy way, while the developer – by using existing computational resources – delivers the application in a short period of time. Furthermore, the registry has to include an abstract layer of resource descriptions and enable development of applications using this layer. The mapping between abstract and actual resources (installed on a remote server) should be made during application execution, with scheduling and optimization algorithms applied to select the best available instance when the application is run.

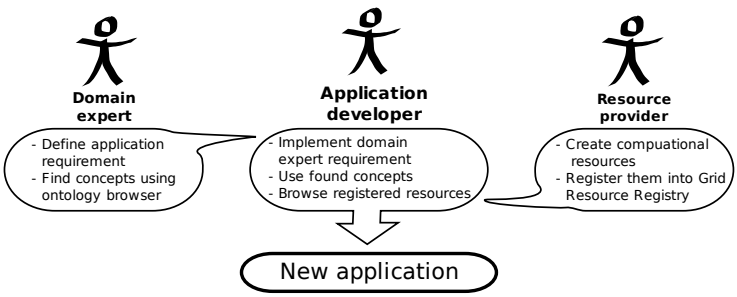

Fig. 1. The process of collaborative e-science application development requires involvement on many actors. We have identified three groups of users. The first one – domain experts – defines application requirements. The second one – application developers – implements the application using resources delivered by third user group – resource providers

The requirements for our system are based on experience from collaboration with modern e-science research teams consisting of three groups of users (see Fig. 1). The first group includes researchers and domain experts who, in most cases, possess limited IT knowledge. This group defines the application requirements for the second group – application developers – skillful enough to actually create the application. Developers use resource available online and write the application glue code, connecting existing pieces of functionality. Actual computational resources have to be delivered by third parties, i.e. our third user group – resource providers. They are the computer scientists able to implement the defined requirements in various technologies.

The identified technologies and user habits allow us to define requirements for the Grid Resource Registry, whose most important features are listed below. These

requirements are split into two groups: the first group deals with simplifying the application development process while the second one is connected with the characteristics of the distributed environment where applications are run. This environment is highly unstable and susceptible to failure, involves significant latencies and topology changes, etc.

From the user perspective, the requirements are as follows:

1. Writing client code for various technologies is often difficult and forces developers to gain knowledge about each technology. Such tasks are time-consuming and error-prone. This problem forces users to define the first requirement for our system: *hiding the technical complexity from the user*, thus allowing him/her to focus on delivering the required application functionality instead of mastering new technologies.
2. Yet another existing issue is connected with online resource documentation or, better to say, the lack of such. This drawback allows us to define the next requirement: *the possibility to annotate* (using human-readable descriptions and ontologies) *each element of a computational resource.*
3. The end user is responsible for defining application requirements. In most cases this user has limited knowledge about computer science, but has broad knowledge about the domain (s)he is an expert in. Basing on this scenario, we have identified another requirement for our system: *the end user should be able to browse the registered resource, using concepts familiar to him/her.*

Additional, environment-related requirements are as follows:

1. The key issue associated with online computational resources is their availability. In a dynamic environment it is often the case that distributed resources become unavailable (e.g. due to routing problems or service restarts). This results in another system requirement: *the ability to find working computational resources during application execution.*
2. Distributed environments deliver many heterogeneous computational resources that can be installed on powerful machines with high network bandwidth, or on average machines with poor bandwidth. It should therefore be possible to match each application to the most suitable services. Additionally, taking into account network changeability, the best service should be found at runtime rather than during development. Thus, *the registry should deliver an optimizer with advanced optimization algorithms, able to locate the best service for the application.*

The requirements presented above are used to define resource descriptions and the architecture of the Grid Resource Registry (GRR). As a result, the registry is a place which connects all three user groups (application end users, developers and resource providers) at each step of the application lifecycle, from application requirements definition to application execution and maintenance.

As stated in Section 2, existing solutions lack a layer of abstraction for describing distributed computational resources, which is a significant drawback from the end user perspective (see Section 3). The abstract layer available in the Grid Resource

Registry addresses this deficiency. The registry reuses patterns available in object-oriented programming, such as interfaces, classes and instances, applying them to descriptions of remote computational resources, created using various technologies (Fig. 2).

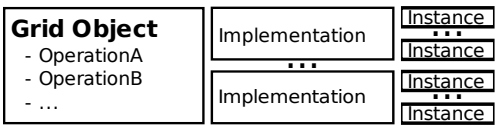

Fig. 2. A resource description consists of three levels. The first level – technology-independent – describes the functionality of the resource while the remaining two technology-specific layers deliver information on how to invoke and locate resources

4. Resource abstraction

The most abstract layer of the resource description is the specification of the resource behavior (*Grid Object*). It describes all service operations, including their input and output parameters. Each parameter, besides a human-readable description, can be assigned an existing ontology concept. Owing to this integration the end user can browse the ontology concept and query all *Grid Objects* and operations which use them. This step reduces the potential for misunderstandings between domain experts (end users) and application developers as they communicate using shared concepts from the same ontology. Additionally, the registry stores information about dependencies between service operations, assisting developers during the application creation phase.

The remaining two layers (*Grid Object Implementation* and *Grid Object Instance*) deal with technical aspect of the service. The *Grid Object Implementation* stores information specific to a given service implementation technology. Furthermore, this layer also allows users to define a nonfunctional description of the service (e.g. its license).

The final layer (*Grid Object Instance*) is used to describe deployment-specific properties, such as endpoints where the service is deployed and installation-related details (e.g. service timeout). This layer is integrated with external subsystems, including monitoring [1] and provenance [2] tools.

5. Grid Resource Registry: architecture and functionality

The central point where information about services is located, is the Grid Resource Registry. It is a module which implements the universal resource description paradigm presented in the previous sections. Fig. 3 presents the registry and its place in the environment for creating and executing applications composed from preexisting remote computational resources. It is used by other system components (and, consequently,

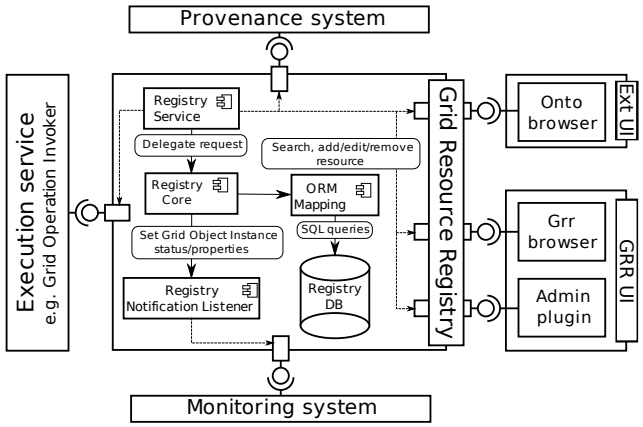

Fig. 3. The Grid Resource Registry supports all three resource description layers. It is integrated with other modules which, together, deliver an environment capable of creating and executing complex distributed applications for e-science

by different groups of users) at each phase of the glue code lifecycle, starting with requirements definition, through development to execution. A detailed description of this architecture is presented below. The registry is split into two parts:

1. The first part is placed on the server side and includes service description storage and registry clients, used by other systems to query, add and edit registry contents.
2. The second part describes required and optional registry user interfaces which assist during the glue code creation process and enable management of registry contents.

5.1. Registry server and its clients

The Grid Resource Registry server is a repository which stores information about computational resources. It is a central place where application developers search for available services, and from which the Execution Service (e.g. the Grid Operation Invoker [14]) receives technical specifications of the services. The Execution Service can ask the repository for services (using its abstract resource description layer), as well as for endpoints of specific service instances. A detailed description of this process is presented in Section 6.

The distributed environment where remote computational resources are installed is highly dynamic. Thus, there is a need to check whether a registered service is working or not. For this purpose the registry is integrated with a monitoring system. Integration is a two-way process. At the beginning the registry notifies the monitoring system about every computational resource which should be monitored. This request contains all technical information required by the monitoring system. The monitoring

system monitors registered services and periodically sends information about service availability (active, inactive, not monitored) to the registry. When the registry receives a message, it sets the appropriate *Grid Object Instance* status. As a result, when the optimizer asks about *Grid Object* instances, only information about working instances is returned.

In choosing the best *Grid Object Instance*, the optimizer [15] takes into account various criteria, including instance reliability. The registry obtains this information thanks to integration with a provenance system. Each time the Execution Service invokes a service operation, it sends an event to the provenance system (informing it whether the invocation was successful or not). The provenance system aggregates this information and periodically calculates service reliability. This information is stored in the Grid Resource Registry and exposed each time the optimizer asks about instances.

Registry browsing is available for all users, but content modification should be restricted to a limited group of users. The Grid Resource Registry supports this functionality by providing web services for content modification, secured using the WS-Security standard. This standard allows us to integrate the registry with various types of security infrastructures (e.g. Shibboleth [11] and GSI).

5.2. User interfaces

Providing a simple method for querying and exploiting registered computational resources is the most important requirement facing the Grid Resource Registry. It cannot be fulfilled without user-friendly interfaces which help the user at each stage of development, starting with application requirements definition, through implementation, to execution.

For the developer, a Grid Resource Registry browser was created. This part of GRR is an Eclipse RPC[1] plugin, which allows users to view and modify the contents of the registry (on every level of resource description – both technology-independent and technology-specific).

Before the developer can begin working on the application, the domain expert has to define the relevant requirements. The Grid Resource Registry bowser provides support for this group of users. It is integrated with the Ontology Browser plugin [5], which allows users to select an ontology class and delegate a query to the GRR browser plugin. Consequently, the user is able to search and browse all *Grid Objects* and operations which use a selected ontology concept. As a result, the domain expert can help the developer search for appropriate services that are going to be used in the application.

[1] `http://wiki.eclipse.org/index.php/Rich_Client_Platform`

6. Example of Grid Resource Registry usage

The process of e-science application development consists of several steps: requirements definition (performed by the domain scientist), glue code development, application release and, finally, application execution.

The first step for every e-science application is identification of the problem domain. For example, the user may require an application for data mining calculations[2]. Subsequently, the application developer searches the ontology for concepts connected with data mining. During this task, the Ontology Browser is used. Integration between the Ontology Browser and the Grid Resource Registry browser plugins ensures that the developer can review all *Grid Objects* and operations which accept or return selected ontology concepts. The developer can either select *Grid Objects* (the technology-independent resource description layer – see Fig. 2) or point to a specific implementation or instance of a given *Grid Object*. If a *Grid Object* is not found, the developer may create a new *Grid Object* and notify the resource provider that an implementation of a particular service is required. In this situation the *Grid Object* is used as an interface for the new service. Once this is done, the resource provider may create (using the most suitable technology) an implementation of the selected *Grid Object*, deploy this *Grid Object Implementation* and register a *Grid Object Instance* in the registry. Now, the application glue code can be completed and released to the end user. Thanks to the abstract resource descriptions delivered by the registry, the developer is not concerned with the specific technology used to create the service instance or with the question which instance will be used during the application execution phase.

A data mining application showcasing our technology and, in particular, the operation of the Grid Resource Registry, is shown in Figure 4. The application code is expressed in JRuby. It spawns two *Grid Objects*. The first one (`WekaGem` – line 1) is responsible for loading data from the database (lines 2–3) and preparing it for data mining calculations (lines 5–7). Additionally, it is responsible for checking classification effectiveness (lines 12–13). The second object (`OneRuleClassifier` – line 9) creates a single-rule classifier, which is trained using sample data (line 10). Subsequently, actual classification is performed (line 11). Proper classification depends on the training operation. This information (dependencies between operations) is stored in the registry and the developer is notified about it during the application development phase.

The most important part of this application involves creating grid objects (lines 1 and 9), where a technology-independent registry layer is used. Thus, the developer does not need to focus on resource implementation technology and placement. The identification of the resource occurs during the execution phase, when the Execution Service connects to the registry and requests the best resource instance, along with

[2] This is a typical example of GRR usage; other examples of successful usage of GRR can be found at `http://virolab.cyfronet.pl`

```
1  retriever =  GObj.create('WekaGem')
2  A = retriever.loadDataFromDatabase(
3            DATABASE, QUERY, USER, PASSWORD)
4
5  B = retriever.splitData(A, 50)
6  trainA = B.trainingData
7  testA = B.testingData
8
9  classifier = GObj.create('OneRuleClassifier')
10 classifier.train(trainA, attributeName)
11 prediction = classifier.classify(testA)
12 classificationPercentage = retriever.compare(
13                   testA, prediction, attributeName)
14
15 puts 'Predicted data: ' + prediction
16 puts 'Prediction quality: ' + classificationPercentage
```

Fig. 4. Simple data mining application presenting the idea of applying resource abstractions, as described in this paper. The application is written in JRuby

its technology-related information. The registry locates all *Grid Object Implementations* and working *Instances* and passes this information to the optimizer. Depending on the optimization algorithm and received information about instances (e.g. service technology, availability, reliability, specification of the machine on which the service instance is installed), the optimizer chooses the most suitable instance for every application run. As a result, the Execution Service receives a technical description of the selected instance which allows it to invoke the resource. The registry is continually updated by external systems, such as provenance and monitoring – thus the data sent to the optimizer is always up-to-date.

7. Summary

This paper presents the idea of describing remote computational resources using three layers: one technology-independent and two technology-specific. This distinction simplifies the process of developing e-science applications allowing developers to focus on delivering the required functionality instead of creating technology-specific application code. This registry facilitates collaboration between three users groups represented in modern research teams. The Grid Resource Registry repository implements the defined resource description paradigm, enabling universal access to computational resources and rapid development of collaborative applications.

The presented resource registry is used in the ViroLab, Gredia and PL-Grid projects to create applications for virology, bioinformatics and chemistry research teams, as well as for media and banking user groups. During these projects we have

verified that the requirements defined in Section 3 are indeed fulfilled. Currently, we have more than thirty production applications released and executed frequently. This demonstrates the universal nature of the Grid Resource Registry. A similar registry will be present in the UrbanFlood project to store information about resources in the Common Information Space[3].

Acknowledgements

This work was partially funded by the European Commission under the UrbanFlood–248767 project. The Authors are grateful to Maciej Malawski and Piotr Nowakowski for their comments and suggestions.

References

[1] Balis B., Bubak M., Łabno B.: *Monitoring of grid scientific workflows.* Sci. Program., 16(2-3), 2008, pp. 205–216.

[2] Balis B., Bubak M., Pelczar M.: *From monitoring data to experiment information – monitoring of grid scientific workflows.* [in:] E-SCIENCE '07: Proc. of the Third IEEE International Conference on e-Science and Grid Computing, Bangalore, India, 2007. IEEE Computer Society, pp. 77–84.

[3] *BioCatalogue*, 2009. `http://www.biocatalogue.org`.

[4] Bubak M., Gubala T., Kapalka M., Malawski M., Rycerz K.: *Workflow composer and service registry for grid applications.* Future Generation Computer Systems, 21(1), January 2005, pp. 79–86.

[5] Bubak M., Malawski M., Gubala T., Kasztelnik M., Nowakowski P., Harezlak D., Bartynski T., Kocot J., Ciepiela E., Funika W., Krol D., Balis B., Assel M., Tirado-Ramos A.: *Handbook of Research on Computational Grid Technologies for Life Sciences, Biomedicine and Healthcare*, chapter Virtual Laboratory for Collaborative Applications. Information Science Reference, IGI Global, 2009.

[6] EMBRACE, 2009. `http://www.embraceregistry.net`.

[7] Foster I., Carl K.: *Scaling system-level science: Scientific exploration and it implications.* Computer, 39(11), November 2006, pp. 31–39.

[8] The Apache Software Funcation. *Apache juddi webpage.* `http://ws.apache.org/juddi/`.

[9] Gil Y., Deelman E., Ellisman M., Fahringer T., Fox G., Gannon D., Goble C., Livny M., Moreau L., Myers J.: *Examining the challenges of scientific workflows.* Computer, 40(12), 2007, pp. 24–32.

[10] Huang Z., Du B., Gu L., He C., Li S.: *Distributed grid resource registry meta-service: design and implementation.* [in:] Autonomous Decentralized Systems – ISADS 2005, 2005, pp. 531–535.

[11] *Internet 2 Project.* Shibboleth, 2008. `http://shibboleth.internet2.edu/`.

[3] http://urbanflood.eu

[12] Kryza B., Słota R., Majewska M., Pieczykolan J., Kitowski J.: *Grid organizational memory-provision of a high-level grid abstraction layer supported by ontology alignment*. Future Gener. Comput. Syst., 23, March 2007, pp. 348–358.

[13] Lord P., Alper P., Wroe C., Goble C.: *Feta: A light-weight architecture for user oriented semantic service discovery*. The Semantic Web: Research and Applications, 2005, pp. 17–31.

[14] Malawski M., Bartyński T., Bubak M.: *Invocation of operations from script-based grid applications*. Future Generation Computer Systems, 26, May 2009, pp. 138–146.

[15] Malawski M., Kocot J., Ryszka I., Bubak M., Wieczorek M., Fahringer T.: *Optimization of application execution in the gridspace environment*. [in:] S. Gorlatch, P. Fragopoulou, and T. Priol, editors, CoreGRID Integration Workshop 2008 – Integrated Research in Grid Computing, April 2008, pp. 395–405.

[16] Nowakowski P., Harezlak D., Bubak M.: *A new approach to development and execution of interactive applications on the grid*. [in:] CCGRID '08: Proceedings of the 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid, Washington, DC, USA, 2008. IEEE Computer Society, pp. 681–686.

[17] Oinn T. M., Greenwood R. M., Addis M., Alpdemir M. N., Ferris J., Glover K., Goble C. A., Goderis A., Hull D., Marvin D., Li P., Lord P. W., Pocock M. R., Senger M., Stevens R., Wipat A., Wroe C.: *Taverna: lessons in creating a workflow environment for the life sciences*. Concurrency and Computation: Practice and Experience, 18(10), 2006, pp. 1067–1100.

[18] OMII-UK. *Grimoires*. `http://www.grimoires.org/`.

[19] OSASIS. *Uddi standard website*, 2007. `http://uddi.xml.org/`.

[20] *Seekda*, 2009. `http://seekda.com`.