MathCloud: Publication and Reuse of Scientific Applications as RESTful Web Services

Alexander Afanasiev, Oleg Sukhoroslov, Vladimir Voloshinov

Institute for Information Transmission Problems of the Russian Academy of Sciences,
Bolshoy Karetny per. 19, Moscow, 127994, Russia
oleg.sukhoroslov@gmail.com

Abstract. The paper presents MathCloud platform which enables wide-scale sharing, publication and reuse of scientific applications as RESTful web services. A unified interface of computational web service based on REST architectural style is proposed. Main components of MathCloud platform including service container, service catalogue, workflow management system, and security mechanism are described. In contrast to other similar efforts based on WS-* specifications, the described platform provides a more lightweight solution with native support for modern Web applications. The platform has been successfully used in several applications from various fields of computational science that confirm the viability of proposed approach and software platform.

Keywords: computational web service, service-oriented scientific environment, software as a service, REST, service container, service catalogue, workflow

1 Introduction

Modern scientific research is closely related to complex computations and analysis of massive datasets. Computational Science is a rapidly growing field that uses advanced computing and data analysis to solve complex scientific and engineering problems. In their research scientists actively use software applications that implement computational algorithms, numerical methods and models of complex systems. Typically, these applications require massive amounts of calculations and are often executed on supercomputers or distributed computing systems.

The increasing complexity of problems being solved requires simultaneous use of several computational codes and computing resources. This leads to an increased complexity of applications and computing infrastructures. The multi- and interdisciplinary nature of modern science requires collaboration within distributed research projects including coordinated use of scientific expertise, software and resources of each partner. This brings a number of problems faced by a scientist in a day-to-day research.

The reuse of existing computational software is one of key factors influencing research productivity. However, the increased complexity of such software means that it often requires specific expertise in order to install, configure and run it that is beyond

the expertise of an ordinary researcher. This specific expertise also involves configuration and use of high performance computing resources required to run the software. In some cases such expertise can be provided by IT support staff, but this brings additional operating expenses that can be prohibitive for small research teams. The problem amplifies in case of actively evolving software which means that it has to be upgraded or reinstalled on a regular basis. In addition to problem-specific parameters many applications require specification of additional runtime parameters such as number of parallel processes. Mastering these parameters also requires additional expertise that sometimes can only be provided by software authors.

Modern supercomputing centers and grid infrastructures provide researchers with access to high performance computing resources. Such facilities also provide access to preinstalled popular computational packages which partially solves the aforementioned problem. Researchers can also use such facilities to run arbitrary computational code. But here lies another problem. In this case, in addition to master the software, the researcher also has to master the subtleties of working with the command line and the batch system of supercomputer or grid middleware. About 30 years ago such interface was taken for granted, but in the eyes of a modern researcher it looks the same as a text web browser - awkward and archaic. Without radically changing their interface, scientific computing facilities have grown, become more complex inside and harder to use.

The third issue faced by a modern computational scientist is related to the need to combine multiple applications such as models or solvers in order to solve a complex problem. Typically, this issue represents a complex problem on its own which naturally includes all the issues discussed previously. It also brings an important problem of interoperability between computational applications written by different authors. Some applications are designed without interoperability in mind which means that this issue has to be resolved by researcher itself.

The described problems severely reduce the research productivity by not allowing scientists to focus on real problems to be solved. Therefore there is a huge demand for high-level interfaces and problem solving environments that hide the complexity of applications and infrastructure from a user.

The most promising approach for taming complexity and enabling reuse of applications is the use of service-oriented architecture (SOA). SOA consists of a set of principles and methodologies for provision of applications in the form of remotely accessible, interoperable services. The use of SOA can enable wide-scale sharing, publication and reuse of scientific applications, as well as automation of scientific tasks and composition of applications into new services [1].

The provision of applications as services is closely related to “Software as a Service” (SaaS) software delivery model implemented nowadays by many web and cloud computing services. This model has several advantages in comparison to traditional software delivery such as ability to run software without installation using a web browser, centralized maintenance and accelerated feature delivery. The ubiquity of SaaS applications and the ability to access these applications via programmable APIs have spawned development of mashups that combine data, presentation and functionality from multiple services, creating a composite service.

A key observation here is that, in essence, the aforementioned issues are not unique to scientific computing. However, it is still an open question how existing approaches, such as SOA, SaaS and Web 2.0, can be efficiently applied in the context of scientific computing environments.

The paper presents MathCloud platform which enables wide-scale sharing, publication and reuse of scientific applications as RESTful web services. Section 2 introduces a unified remote interface of computational web service based on REST architectural style. Section 3 describes main components of MathCloud platform including service container, service catalogue, workflow management system, and security mechanism. Section 4 presents applications and experimental evaluation of created platform. Section 5 discusses related work.

2 Unified Interface of Computational Web Service

Currently, the dominant technology for building service-oriented systems are Web services based on SOAP protocol, WSDL and numerous WS-* specifications (hereinafter referred to as "big Web services"). A common criticism of big Web services is their excessive complexity and incorrect use of core principles of the Web architecture [2]. The advantages of big Web services mostly apply to complex application integration scenarios and business processes that occur in enterprise systems, while rarely present in Web 2.0 applications that favor ease-of-use and ad hoc integration [3].

The most promising alternative approach to implementation of web services is based on the REST (Representational State Transfer) architectural style [4]. Thanks to the uniform interface for accessing resources, the use of core Web standards and the presence of numerous proven implementations, REST provides a lightweight and robust framework for development of web services and related client applications. This is confirmed by a proliferation of the so-called RESTful web services [2], especially within Web 2.0 applications.

Assuming that service-oriented scientific environments should also emphasize ease-of-use and ad hoc integration of services, we propose to implement computational services as RESTful web services with a unified interface [5]. This interface or REST API is based on the following abstract model of a computational service. A service processes incoming client's requests to solve specific problems. A client's request includes a parameterized description of the problem, which is represented as a set of input parameters. Having successfully processed the request, the service returns the result represented as a set of output parameters to the client.

The proposed unified interface of computational web service is formed by a set of resources identified by URIs and accessible via standard HTTP methods (Table. 1). The interface takes into account features of computational services by supporting asynchronous request processing and passing large data parameters. Also, in accordance with the service-oriented approach, the interface supports introspection, i.e., obtaining information about the service and its parameters.

Table 1. REST API of computational web service.

Resource	GET	POST	DELETE
Service	Get service description	Submit new request (create job)	
Job	Get job status and results		Cancel job, delete job data
File	Get file data		

The service resource supports two HTTP methods. GET method returns the service description. POST method allows a client to submit a request to server. The request body contains values of input parameters. Some of these values may contain identifiers of file resources. In response to the request, the service creates a new subordinate job resource and returns to the client identifier and current representation of the job resource.

The job resource supports GET and DELETE methods. GET method returns job representation with information about current job status. If the job is completed successfully, then the job representation also contains job results in the form of values of output parameters. Some of these values may contain identifiers of file resources.

The DELETE method of job resource allows a client to cancel job execution or, if the job is already completed, delete job results. This method destroys the job resource and its subordinate file resources.

The file resource represents a part of client request or job result provided as a remote file. The file contents can be retrieved fully or partially via the GET method of HTTP or other data transfer protocol.

Note that the described interface doesn't prescribe specific templates for resource URIs which may vary between implementations. It is desirable to respect the described hierarchical relationships between resources while constructing these URIs.

The described interface supports job processing in both synchronous and asynchronous modes. Indeed, if the job result can be immediately returned to the client, then it is transmitted inside the returned job resource representation along with the indication of DONE state. If, however, the processing of request takes time, it is stated in the returned job resource representation by specifying the appropriate job state (WAITING or RUNNING). In this case, the client uses the obtained job resource identifier for further checking of job state and obtaining its results.

The proposed REST API is incomplete without considering resource representation formats and means of describing service parameters. The most widely used data representation formats for Web services are XML and JSON. Among these JSON has been chosen for the following reasons. First, JSON provides more compact and readable representation of data structures, while XML is focused on representation of arbitrary documents. Second, JSON supports native integration with JavaScript language simplifying creation of modern Ajax based Web applications.

A known disadvantage of JSON is the lack of standard tools for description and validation of JSON data structures comparable to XML Schema. However, there is an active ongoing work on such format called JSON Schema [6]. This format is used for description of input and output parameters of computational web services within the proposed REST API.

3 MathCloud Platform

MathCloud platform [7] is a software toolkit for building, deployment, discovery and composition of computational web services using the proposed REST API. This section presents main components of MathCloud platform.

3.1 Service Container

Service container codenamed Everest represents a core component of the platform. Its main purpose is to provide a high-level framework for development and deployment of computational web services. Everest simplifies service development by means of ready-to-use adapters for common types of applications. The container also implements a universal runtime environment for such services based on the proposed REST API. The architecture of Everest is presented in Fig. 1.

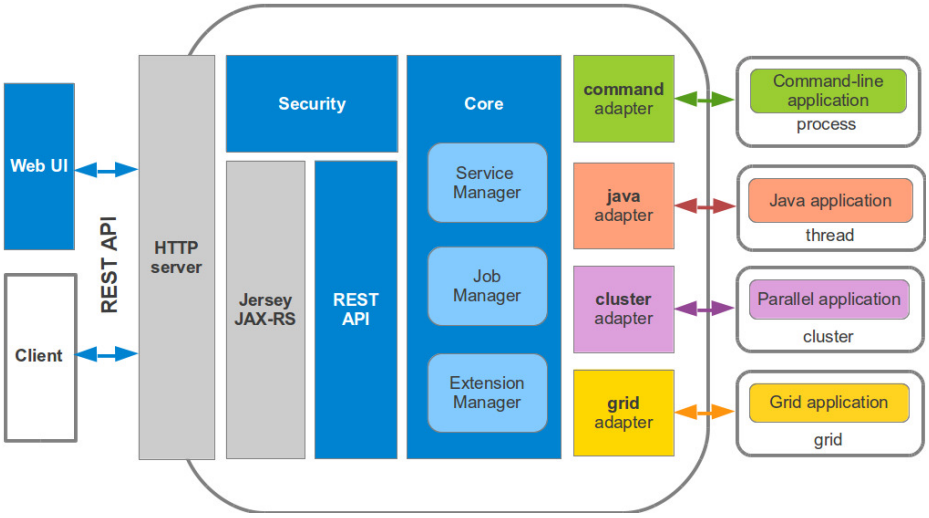

Fig. 1. Architecture of service container

The service container is based on Jersey library, a reference implementation of JAX-RS (Java API for RESTful Web Services) specification. The container uses built-in Jetty web server for interaction with service clients. Incoming HTTP requests are dispatched to Jersey and then to the container. Everest is processing client requests in accordance with configuration information.

The Service Manager component maintains a list of services deployed in the container and their configuration. This information is read at startup from configuration files. The configuration of each service consists of two parts:

- Public service description which is provided to service clients;
- Internal service configuration which is used during request processing.

The Job Manager component manages the processing of incoming requests. The requests are converted into asynchronous jobs and placed in a queue served by a configurable pool of handler threads. During job processing, handler thread invokes adapter specified in the service configuration.

The components that implement processing of service requests (jobs) are provided in the form of pluggable adapters. Each adapter implements a standard interface through which the container passes request parameters, monitors the job state and receives results.

Currently the following universal adapters are implemented.

The Command adapter converts service request to an execution of specified command in a separate process. The internal service configuration contains the command to execute and information about mappings between service parameters and command line arguments or external files.

The Java adapter performs invocation of a specified Java class inside the current Java virtual machine, passing request parameters inside the call. The specified class must implement standard Java interface. The internal service configuration includes the name of the corresponding class.

The Cluster adapter performs translation of service request into a batch job submitted to computing cluster via TORQUE resource manager. The internal service configuration contains the path to the batch job file and information about mappings between service parameters and job arguments or files.

The Grid adapter performs translation of service request into a grid job submitted to the European Grid Infrastructure, which is based on gLite middleware. This adapter can be used both to convert existing grid application to service and to port existing service implementation to the grid. The internal service configuration contains the name of grid virtual organization, the path to the grid job description file and information about mappings between service parameters and job arguments or files.

Note that the all adapters, except Java, support converting of existing applications to services by writing only a service configuration file, i.e., without writing a code. This feature makes it possible for unskilled users to publish as services a wide range of existing applications. Besides that, the support for pluggable adapters allows one to attach arbitrary service implementations and computing resources.

Each service deployed in Everest is published via the proposed REST API. In addition to this, container automatically generates a complementary web interface allowing users to access the service via a web browser.

3.2 Service Catalogue

The main purpose of service catalogue is to support discovery, monitoring and annotation of computational web services. It is implemented as a web application with interface and functionality similar to modern search engines.

After the service is deployed in the service container it can be published in the catalogue by proving a URI of the service and a few tags describing it. The catalogue retrieves service description via the unified REST API, performs indexing and stores description along with specified tags in a database.

The catalogue provides a search query interface with optional filters. It supports full text search in service descriptions and tags. Search results consist of short snippets of each found service with highlighted query terms and a link to full service description.

In order to provide current information on service availability the catalogue periodically pings published services. If a service is not available it is marked accordingly in search results. The catalogue also implements some experimental features similar to collaborative Web 2.0 sites, e.g., ability to tag services by users.

3.3 Workflow Management System

In order to simplify composition of services a workflow management system is implemented [8]. The system supports description, storage, publication and execution of workflows composed of multiple services. Workflows are represented as directed acyclic graphs and described by means of a visual editor. The described workflow can be published as a new composite service and then executed by sending request to this service. The system has client-server architecture. The client part of the system is represented by workflow editor, while the server part is represented by the workflow management service.

Fig. 2 shows the interface of the workflow editor. It is inspired by Yahoo! Pipes and implemented as a Web application in JavaScript language. This makes it possible to use the editor on any computer running a modern web browser.

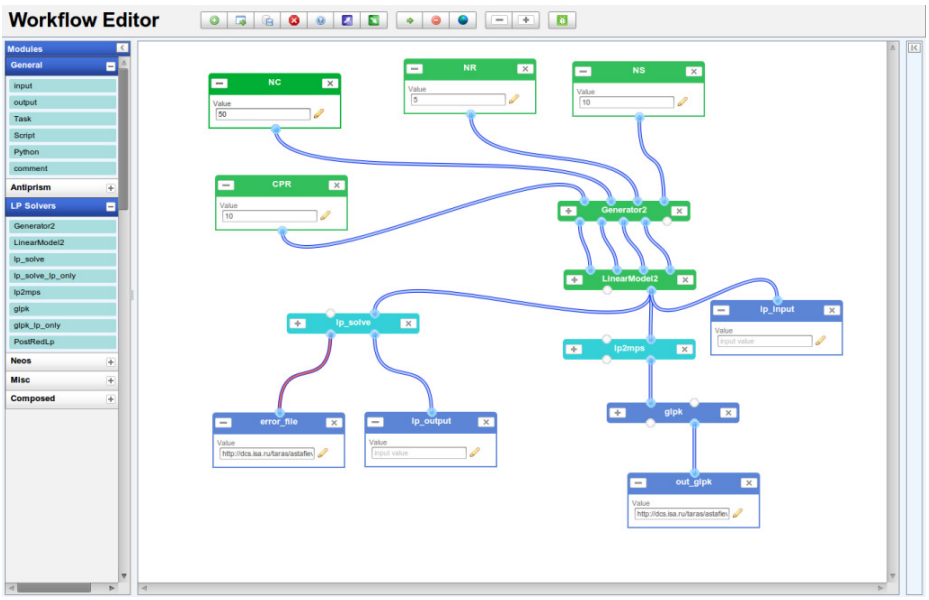

Fig. 2. Graphical workflow editor

The workflow is represented in the form of a directed acyclic graph whose vertices correspond to workflow blocks and edges define data flow between the blocks. Each block has a set of inputs and outputs displayed in the form of ports at the top and at the bottom of the block respectively. Each block implements a certain logic of processing of input data and generating of output data. Data transfer between blocks is realized by connecting the output of one block to the input of another block. Each input or output has associated data type. The compatibility of data types is checked during connecting the ports.

The introduction of a service in a workflow is implemented by creating a new Service block and specifying the service URI. It is assumed that the service implements the unified REST API. This allows the editor to dynamically retrieve service description and extract information about the number, types and names of input and output parameters of the service. This information is used to automatically generate the corresponding input and output ports of the block.

The unified REST API provides a basis for service interoperability on the interface level. The user can connect any output of one service with any input of another service if both ports have compatible data types. However, it is important to note that the system doesn't check the compatibility of data formats and semantics of the corresponding parameters. It is the task of the user to ensure this.

An important feature of the editor is ability to run a workflow and display its state during the execution. Before the workflow can be run it is necessary to set the values of all input parameters of the workflow via the appropriate Input blocks. After the user clicks on the Run button, the editor makes a call with the specified input parameters to the composite service representing the workflow. Then the editor performs a periodic check of the status of running job, which includes information about states of individual blocks of the workflow. This information is displayed to the user by painting each workflow block in the color corresponding to its current state. After successful completion of the workflow, the values of workflow output parameters are displayed in the Output blocks. Each workflow instance has a unique URI which can be used to open the current state of the instance in the editor at any time. This feature is especially useful for long-running workflows.

The workflow management service (WMS) performs storage, deployment and execution of workflows created with the described editor. In accordance with the service-oriented approach the WMS deploys each saved workflow as a new service. The subsequent workflow execution is performed by sending request to the new composite service through the unified REST API. Such requests are processed by the workflow runtime embedded in WMS. The WMS is implemented as a RESTful web service. This provides a most convenient way to interact with the WMS from the workflow editor.

3.4 Security

All platform components use common security mechanism (Fig. 3) for protecting access to services. It supports authentication, authorization and a limited form of delegation based on common security technologies.

Authentication of services is implemented by means of SSL server certificates. Authentication of clients is implemented via two mechanisms. The first one is standard X.509 client certificate. The second is Loginza service which supports authentication via popular identity providers (Google, Facebook, etc.) or any OpenID provider. The latter mechanism, which is available only for browser clients, is convenient for users who don't have a certificate.

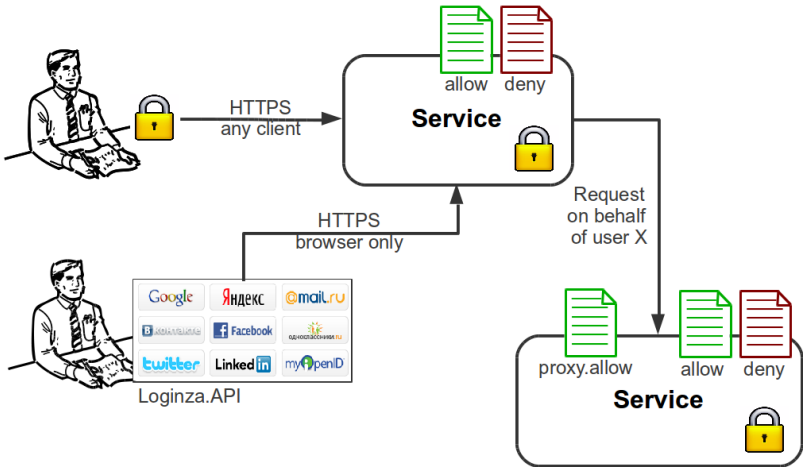

Fig. 3. Security mechanism

Authorization is supported by means of allow and deny lists which enable service administrator to specify users which should or should not have access to a service. A user can be specified using its certificate's distinguished name or OpenId identifier.

A well-known security challenge in service-oriented environments is providing a mechanism for a service to act on behalf of a user, i.e., invoke other services. A common use case is a workflow service which needs to access services involved in the workflow on behalf of a user invoked the service. For such cases a proxying mechanism is implemented by means of a proxy list which enable service administrator to specify certificates of services that are trusted to invoke the service on behalf of users. In comparison to proxy certificate mechanism used in grids, this approach is more limited but provides a lightweight solution compatible with the proposed REST API.

3.5 Clients

The platform provides Java, Python and command-line clients for accessing services from external applications and implementing complex workflows. Since the access to services is implemented via REST API, one can use any standard HTTP library or client (e.g., curl) to interact with services. Also, thanks to using JSON format, services can be easily accessed from JavaScript applications via Ajax calls. This simplifies development of modern Web-based interfaces to services, in contrast to approaches based on big Web services and XML data format.

4 Applications

MathCloud platform has been used in several applications from various fields of computational science. This section describes some of these applications and summarizes general conclusions drawn from their development.

One of the first applications of MathCloud platform concerns an "error-free" inversion of ill-conditioned matrix [9], a well-known challenging task in computational science. The application uses symbolic computation techniques available in computer algebra systems (CAS) which require substantial computing time and memory. To address this issue a distributed algorithm of matrix inversion has been implemented via Maxima CAS system exposed as a computational web service. The algorithm was implemented as a workflow based on block decomposition of input matrix and Schur complement. The approach has been validated by inversion of Hilbert matrices up to 500×500.

The matrix inversion application provided an opportunity to evaluate performance of all platform components, in particular with respect to passing large amounts of data between services. In the case of extremely ill-conditioned matrices the symbolic representation of final and intermediate results reached up to hundreds of megabytes. Table 2 presents obtained performance results including serial execution time in Maxima, parallel execution time in MathCloud (using 4-block decomposition) and observed speedup. Additional analysis revealed that the overhead introduced by the platform including data transfer is about 2-5% of total computing time.

Table 2. Performance of Hilbert (NxN) matrix inversion application in MathCloud.

N	Serial execution time in Maxima, minutes	Parallel execution time in MathCloud, minutes	Speedup
250	8	5	1,60
300	15	8	1,88
350	27	13	2,08
400	45	20	2,25
450	72	30	2,40
500	109	40	2,73

Another MathCloud application has been developed for interpreting the data of X-ray diffractometry of carbonaceous films by means of solving optimization problems within a broad class of carbon nanostructures [10]. The application is implemented as a workflow which combines parallel calculations of scattering curves for individual nanostructures (performed by a grid application) with subsequent solution of optimization problems (performed by three different solvers running on a cluster) to determine the most probable topological and size distribution of nanostructures. All these parts of computing scheme (and a number of additional steps, e.g., data preparation, post-optimal processing and plotting) have been implemented as

computational web services. The application helped to reveal the prevalence of low-aspect-ratio toroids in tested films [11].

A recent work [12-13] concerns a uniform approach to creation of computational web services related to optimization modeling. MathCloud platform is used within this work to integrate various optimization solvers intended for basic classes of mathematical programming problems and translators of AMPL optimization modeling language. A number of created computational web services and workflows cover all basic phases of optimization modeling techniques: input of optimization problems' data, interaction with solvers, processing of solutions found.

A special service has been developed that implements dispatching of optimization tasks to a pool of solver services directly via AMPL translator's execution. These features enable running any optimization algorithm written as an AMPL script in distributed mode when all problems (and/or intermediate subproblems) are solved by remote optimization services. Independent problems are solved in parallel thus increasing overall performance in accordance with the number of available services. The proposed approach has been validated by the example of Dantzig–Wolfe decomposition algorithm for multi-commodity transportation problem.

The experience gained from application development shows that MathCloud platform can be efficiently applied for solving a wide class of problems. This class can be described as problems that allow decomposition into several coarse-grained suproblems (dependent or independent) that can be solved by existing applications represented as services. It is important to note that while MathCloud can be used as a parallel computing platform in homogeneous environments such as a cluster, it is generally not as efficient in this setting as dedicated technologies such as MPI. The main benefits of MathCloud are revealed in heterogeneous distributed environments involving multiple resources and applications belonging to different users and organizations.

The exposing of computational applications as web services is rather straightforward with MathCloud. From our experience it usually takes from tens of minutes to a couple of hours to produce a new service including service deployment and debugging. This is mainly due to the fact that the service interface is fixed and the service container provides a framework that implements all problem-independent parts of a service. That means that a user doesn't need to develop a service from scratch as it happens when using general purpose service-oriented platforms. In many cases service development reduces to writing a service configuration file. In other cases a development of additional application wrapper is needed which is usually accomplished by writing a simple shell or Python script.

The workflow development is somewhat harder, especially in the case of complex workflows. However, the workflow editor provides some means for dealing with this. First of all, it enables dividing complex workflow into several simpler sub-workflows by supporting publishing and composing of workflows as services. Second, it is possible to add custom workflow actions written in JavaScript or Python, for example to create complex string inputs for services from user data or to get additional timing. Finally, besides the graphical editor it is possible to download workflow in JSON format, edit it manually and upload back to WMS. These and other features provide rather good usability for practical use of MathCloud platform.

5 Related Work

The use of service-oriented approach in the context of scientific computing was proposed in [1]. Service-Oriented Science as introduced by Foster refers to scientific research enabled by distributed networks of interoperating services.

The first attempts to provide a software platform for Service-Oriented Science were made in Globus Toolkit 3 and 4 based on the Open Grid Services Architecture (OGSA) [14]. OGSA describes a service-oriented grid computing environment based on big Web services. Globus Toolkit 3/4 provided service containers for deployment of stateful grid services that extended big Web services. These extensions were documented in the Web Services Resource Framework (WSRF) specification [15]. It largely failed due to inherent complexity and inefficiencies of both specification and its implementations. Globus Toolkit 4 had steep learning curve and provided no tools for rapid deployment of existing applications as services and connecting services to grid resources.

There have been several efforts aiming at simplifying transformation of scientific applications into remotely accessible services. The Java CoG Kit [16] provided a way to expose legacy applications as Web services. It uses a serviceMap document to generate source code and WSDL descriptor for the Web service implementation. Generic Factory Service (GFac) [17] provides automatic service generation using an XML-based application description language. Instead of source code generation, it uses an XSUL Message Processor to intercept the SOAP calls and route it to a generic class that invokes the scientific application. SoapLab [18] is another toolkit that uses an application description language called ACD to provide automatic Web service wrappers.

Grid Execution Management for Legacy Code Architecture (GEMLCA) [19] implements a general architecture for deploying legacy applications as grid services. It implements an application repository and a set of WSRF-based grid services for deployment, execution and administration of applications. Instead of generation of different WSDLs for every deployed application as in GFac and SoapLab, GEMLCA uses generic interface and client for application execution. The execution of applications is implemented by submission of grid jobs through back-end plugins supporting Globus Toolkit and gLite middleware. GEMLCA was integrated with the P-GRADE grid portal [20] in order to provide user-friendly Web interfaces for application deployment and execution. The workflow editor of the P-GRADE portal supports connection of GEMLCA services into workflows using a Web-based graphical environment.

Opal [21] and Opal2 [22] toolkits provide a mechanism to deploy scientific applications as Web services with standard WSDL interface. This interface provides operations for job launch (which accepts command-line arguments and input files as its parameters), querying status, and retrieving outputs. In comparison to GEMLCA, Opal toolkit deploys a new Web service for each wrapped application. Opal also provides an optional XML-based specification for command-line arguments, which is used to generate automatic Web forms for service invocation. In addition to Base64 encoded inputs, Opal2 supports transfer of input files from remote URLs and via

MIME attachments which greatly improves the performance of input staging. It supports several computational back-ends including GRAM, DRMAA, Torque, Condor and CSF meta-scheduler.

The described toolkits have many similarities with the presented software platform, e.g., declarative application description, uniform service interface, asynchronous job processing. A key difference is related to the way services are implemented. While all mentioned toolkits use big Web services and XML data format, the MathCloud platform exposes applications as RESTful web services using JSON format. The major advantages of this approach are decreased complexity, use of core Web standards, wide adoption and native support for modern Web applications as discussed in Section 2.

The idea of using RESTful web services and Web 2.0 technologies as a lightweight alternative to big Web services for building service-oriented scientific environments was introduced in [23]. Given the level of adoption of Web 2.0 technologies relative to grid technologies, Fox et al. suggested the replacement of many grid components with their Web 2.0 equivalents. Nevertheless, to the authors' knowledge, there are no other efforts to create a general purpose service-oriented toolkit for scientific applications based on RESTful web services.

There are many examples of applying Web 2.0 technologies in scientific research in the form of Web-based scientific gateways and collaborative environments [20, 24-25]. While such systems support convenient access to scientific applications via Web interfaces, they don't expose applications as services thus limiting application reuse and composition.

There are many scientific workflow systems, e.g. [26-28]. The system described in the paper stands out among these by providing a Web-based interface, automatic publication of workflows as composite services and native support for RESTful web services.

6 Conclusion

The paper presented MathCloud platform which enables wide-scale sharing, publication and reuse of scientific applications as RESTful web services based on the proposed unified REST API. In contrast to other similar efforts based on Web Services specifications, it provides a more lightweight solution with native support for modern Web applications. MathCloud includes all core tools for building a service-oriented environment such as service container, service catalogue and workflow system. The platform has been successfully used in several applications from various fields of computational science that confirm the viability of proposed approach and software platform.

The future work will be focused on building a hosted Platform-as-a-Service (PaaS) for development, sharing and integration of computational web services based on the described software platform.

Acknowledgements. The work is supported by the Presidium of the Russian Academy of Sciences (the basic research program No.14) and the Russian Foundation for Basic Research (grant No. 11-07-00543-a).

References

1. Foster, I.: Service-Oriented Science. Science, vol. 308, no. 5723, pp. 814–817 (2005)
2. Richardson, L., Ruby, S.: RESTful Web Services. O'Reilly Media (2007)
3. Pautasso, C., Zimmermann, O., Leymann, F.: Restful web services vs. "big"' web services: making the right architectural decision. In: 17th international conference on World Wide Web (WWW '08). ACM, New York, NY, USA, pp. 805-814 (2008)
4. Fielding, R. T.: Architectural Styles and the Design of Network-based Software Architectures. Ph.D. dissertation, University of California, Irvine, Irvine, California (2000)
5. Sukhoroslov, O.V.: Unified Interface for Accessing Algorithmic Services in Web. Proceedings of ISA RAS, vol. 46, pp. 60-82 (2009) (in Russian)
6. JSON Schema, `http://www.json-schema.org/`
7. MathCloud Project, `http://mathcloud.org/`
8. Lazarev, I.V., Sukhoroslov, O.V.: Implementation of Distributed Computing Workflows in MathCloud Environment. Proceedings of ISA RAS, vol. 46, pp. 6-23 (2009) (in Russian)
9. Voloshinov, V.V., Smirnov, S.A.: Error-Free Inversion of Ill-Conditioned Matrices in Distributed Computing System of RESTful Services of Computer Algebra. In: 4th Intern. Conf. Distributed Computing and Grid-Technologies in Science and Education, pp. 257-263. JINR, Dubna (2010)
10. Vneverov, V.S., Kukushkin, A.B., Marusov, N.L. et al.: Numerical Modeling of Interference Effects of X-Ray Scattering by Carbon Nanostructures in the Deposited Films from TOKAMAK T-10. Problems of Atomic Science and Technology, Ser. Thermonuclear Fusion, Vol. 1, 2011, pp. 13-24 (2011) (in Russian)
11. Kukushkin, A.B., Neverov, V.S., Marusov, N.L. et al.: Few-nanometer-wide carbon toroids in the hydrocarbon films deposited in tokamak T-10. Chemical Physics Letters 506, pp. 265–268 (2011)
12. Voloshinov, V.V., Smirnov, S.A.: On development of distributed optimization modelling systems in the REST architectural style. In: 5th Intern. Conf. Distributed Computing and Grid-Technologies in Science and Education. JINR, Dubna (2012)
13. V.V. Voloshinov, S.A. Smirnov: Software Integration in Scientific Computing. Information Technologies and Computing Systems, No. 3, pp. 66-71 (2012) (in Russian)
14. Foster, I., Kesselman, C., Nick, J., Tuecke, S.: Grid Services for Distributed System Integration. Computer 35(6), pp. 37-46 (2002)
15. WS-Resource Framework, `http://www-106.ibm.com/developerworks/library/ws-resource/ws-wsrf.pdf`
16. von Laszewski, G., Gawor, J., Krishnan, S., Jackson, K.: Commodity Grid Kits - Middleware for Building Grid Computing Environments. In: Grid Computing: Making the Global Infrastructure a Reality, chapter 25. Wiley (2003)
17. Kandaswamy, G., Fang, L., Huang, Y., Shirasuna, S., Marru, S., Gannon, D.: Building Web Services for Scientific Grid Applications. IBM Journal of Research and Development, , vol. 50, no. 2.3, pp. 249-260 (2006)
18. SoapLab Web Services, `http://www.ebi.ac.uk/soaplab/`

19. Delaitre, T., Kiss, T., Goyeneche, A., Terstyanszky, G., Winter, S., Kacsuk, P.: GEMLCA: Running Legacy Code Applications as Grid Services. Journal of Grid Computing Vol. 3. No. 1-2, pp. 75-90 (2005)
20. Kacsuk, P., Sipos, G.: Multi-Grid, Multi-User Workflows in the P-GRADE Portal. Journal of Grid Computing, Vol. 3, No. 3-4, Springer, pp. 221-238 (2005)
21. Krishnan, S., Stearn, B., Bhatia, K., Baldridge, K. K., Li, W., Arzberger, P.: Opal: Simple Web Services Wrappers for Scientific Applications. In: IEEE Intl. Conf. on Web Services (ICWS) (2006)
22. Krishnan, S., Clementi, L., Ren, J., Papadopoulos, P., Li, W.: Design and Evaluation of Opal2: A Toolkit for Scientific Software as a Service. In: 2009 IEEE Congress on Services (SERVICES-1 2009), pp.709-716 (2009)
23. Fox, G., Pierce, M.: Grids Challenged by a Web 2.0 and Multicore Sandwich. Concurrency and Computation: Practice and Experience, vol. 21, no. 3, pp. 265–280 (2009)
24. McLennan, M., Kennell, R.: HUBzero: A Platform for Dissemination and Collaboration in Computational Science and Engineering. Computing in Science and Engineering, 12(2), pp. 48-52, March/April (2010)
25. Afgan, E., Goecks, J., Baker, D., Coraor, N., Nekrutenko, A., Taylor, J.: Galaxy - a Gateway to Tools in e-Science. In: K. Yang, Ed. (ed) Guide to e-Science: Next Generation Scientific Research and Discovery, pp. 145-177. Springer (2011)
26. Altintas, I., Berkley, C., Jaeger, E., Jones, M., Ludäscher, B., Mock, S.: Kepler: an extensible system for design and execution of scientific workflows. In: Proceedings of the 16th International Conference on Scientific and Statistical Database Management (SSDBM '04), pp. 423-424 (2004)
27. Deelman, E., Singh, G., Su, M. H. et al.: Pegasus: a framework for mapping complex scientific workflows onto distributed systems. Scientific Programming, vol. 13, no. 3, pp. 219-237 (2005)
28. Oinn, T., Greenwood, M., Addis, M. et al.: Taverna: lessons in creating a workflow environment for the life sciences. Concurrency Computation Practice and Experience, vol. 18, no. 10, pp. 1067–1100 (2006)