An Approach to Problem-Oriented Interfaces for Applications in Distributed Computing Systems

Aleksei A. Sorokin¹, Sergey P. Korolev¹, Sergey I. Malkovsky¹, Alexei G. Tarasov¹, and Konstantin V. Mikhailov¹

Computing Center of the Far Eastern Branch, Russian Academy of Sciences, Khabarovsk, Russia
alsor@febras.net

Abstract
The paper describes the implementation of an efficient user interface for applied software in distributed computing systems. As part of this work, a software platform has been developed for the formation of problem-oriented interfaces for application software packages and components of distributed computing systems. The main purpose of the system is to simplify the input of technical settings, launch specific jobs if required and monitor all intermediate processes. Thus, it provides an intuitive user interface for working directly with an application system, and does not require knowledge of low-level operations. Using web services and modern information technology, the developed platform makes it possible to create a set of simple and convenient interfaces for researchers who use high-performance computer systems in their work. As an example, we describe the adaptation of the FHI98md code for a Grid system.

Keywords: distributed computing systems, problem-oriented interface, database, Representational State Transfer, JavaScript Object Notation, application programming interface, web services, Grid, FHI98md.

1 Introduction
A large number of applied software and information technologies capable of having a significant impact on the quality and speed of ongoing researches are nowadays eliminated or used at minimal capacity. One reason is that the complexity of interaction with computer systems increases due to higher levels of performance. The other reason is because the specificity of HPC does not allow professionals in subject fields far from computing technologies to quickly resolve any software and hardware issues.

One solution to these issues is to hide all technical details of the computing tasks from the user and, instead, provide an intuitive user interface that does not require low-level operations.

¹Corresponding author. Tel.: +7-421-270-3913 fax: +7-421-222-7267.
2 Statement of the problem

Currently, considerable attention is focused on the development of HPC software systems and the improvement of their operations. It is worth mentioning here that international consortia are designing tools for Grid implementations [7] (e.g., Globus [6], parallel computing scheduling package Condor-G [9] and X-COM system [11]). Many other solutions, based on different hardware platforms and effectively integrating distributed computing nodes into a single managed system with unified access to resources, also become available. Nevertheless, only a limited number of studies are aimed at simplifying the end-user interaction with computing systems [12]. For example, the Russian Grid NNN project [5] uses a single information portal to implement the application programming interface (API), which allows the addition of new Grid application suites. In spite of the advantages of this project, its interaction with the end user has the following disadvantages:

1. The API is quite restrictive. Because of the specificity of the portal, one cannot create user interfaces that go beyond the provided API. This is well illustrated by the following examples:
   - The API is located on the portal server, so users require (unwanted) access rights to this server to modify the library and load extension modules (plugins);
   - The API uses an existing implementation of a JavaScript library, which makes it difficult to create automated problem-oriented interfaces (POI) for computing jobs outside this environment with AJAX support.

2. The API combines two separate sets of functions that implement both the creation of configuration files and submission of computing jobs for execution. This complicates the end user work dramatically, because a wide range of technical operations must be performed (data entry, setting job related parameters, downloading the generated files to their workstation, creating and submitting the job alongside with a set of the corresponding configuration files to the server). This can lead to errors and resubmitting of the job.

3. A significant part of the API functions is handed to the user interface, potentially compromising the system operations by substituting client side plugins.

A similar approach is used in existing solutions of manufacturers of computing systems and management software for them. These include Moab HPC Suite Application Portal Edition [1] from Adaptive Computing, Altair Compute Manager [4] from Altair Engineering and Platform Application Center [8] from IBM. The applicability of above solutions to heterogeneous computing resources management is hampered by their binding to specific computer systems or to specific control software. In addition, the opportunity of modifying such information systems for performance of specific tasks is significantly limited by provided APIs and lack of access to the source code.

The most common approach to solving the above listed problems of software adaptation is to provide a user interface for a specific application system. For example, the "ABINIT
on nanoHUB project is an adaptation of the applied software package (ASP) ABINIT [13] for quantum-mechanical calculations. Another example which illustrates this approach can be called the web interface HydroGate [2], designed to work with TauDEM software (hydrologic terrain analysis tools).

The aforementioned approaches have advantages and disadvantages. They are tightly coupled to either the application or the computing system itself. This hampers the flexible and efficient generation of easy-to-use interfaces for subject area specialist in the Grid or on standalone computing clusters.

To isolate the server and client parts of user interface development, a unified software platform has been developed (Figure 1) providing interaction between the components of a distributed computing system (DCS) and ASPs. The proposed software platform design consists of two major components, a server and a web-based front-end. These communicate via the RESTful protocol, with control commands represented in JavaScript object notation (JSON). This approach enables to independently modify the software implementations of each of the components.

![Figure 1: Developed platform architecture.](image)

### 3 Software Platform Design

The server component implements the necessary adapters to toolkits for computing environment management. The platform then uses command-line utilities available as part of Pilot-CLI or OpenSSL and, if available, an appropriate instrumentation API. For example, to support PBS Torque and Pilot-CLI built using the Globus Toolkit, several software adapters were developed to query the state of computing resources, submit jobs for execution and so on. Thus, the two DCS toolkits most widely used in the scientific and educational environment are supported.

The following server component modules were created:

- authorization of the platform components and users of the system;
- adapters to convert user requests to specific requests supported by the computing environment;
• input parameter validation and building configuration and auxiliary files for software packages based on a set of rules and templates;

• file management, providing information on the state of computing jobs;

• job management with the preservation of unique job identifier assignments, regardless of the computing environment;

• support for testing the developed software platform.

The server component provides control to interact with the client-side component of the platform. Request handling (accepting client connections, dispatching incoming requests) is performed by the Tornado framework or any other server that supports the web server gateway interface (WSGI, www.wsgi.org).

The web-based components are built with WSGI, which processes queries received from the Apache web server with support for SSL-enabled connections.

The component includes implementations for the following functions:

• user authorization;

• generation of web forms, representing a set of related parameters depending on the specific subject area;

• viewing text files and downloading binary files containing calculation results;

• providing an interface to control and manage the status of computing jobs;

• basic functions for mapping the data model to the web interface (e.g. HTML page layout, JavaScript bindings).

The software platform components were implemented in Python using the DBMS MySQL, ORM SQLAlchemy and a number of standard system libraries.

The user authorization in the system is implemented using x.509 certificates. This allows integration of the developed software infrastructure into already-established grid systems. In addition, x.509 certificates were implemented using GOST (Russian standards) certificates, which matched to trusted x.509 counterparts by the Distinguished Name field. This approach facilitates approval of DCS by the relevant Russian standards certificates.

The Platform Grid module provides job submission running on behalf of the server component using a single certificate assigned to the platform, and submission using the MyProxy service and a certificate delegated by the user. In both cases, the job is processed by the platform, regardless of any jobs submitted by other users.

Creating a managed file subsystem allows hiding implementation details and the physical structure/hierarchy of files from user. This increases the reliability of the platform, preventing intentional and unintentional access based on the assumption of file location. Each file contains meta-information regarding the particular computing job, system and subsystem that owns/created the file.

To create job configuration files, multiple file templates are used. Based on input parameters, they form final files that are submitted for execution (e.g., a JDSL description file) and configuration of the target application (e.g. an .ini file with exact problem parameters). This general approach is common to all systems based on POIs. The advantage of the developed platform is its high flexibility and the automated creation and description of the job input files.
The parameters themselves are described in the files, along with the parameter validation conditions. This simplifies the addition of appropriate POIs without reconfiguring the source code of the platform.

All generated files can be divided into two categories: those related to the job (e.g., description of the required resources in a computing environment), and those related to the problem (e.g., configuration files of a particular software package). This provides a simple means of transferring the same computational jobs among various scheduling systems to various computing environments (e.g., Grid, computing cluster job-queue manager).

The job lifecycle management module [3] assigns a unique internal identifier for each job that is used by specific tools to build and maintain the computing environment. A similar mechanism is used in Grid systems, where a unique identifier hides the identifiers assigned by the batch systems in specific clusters. In the developed platform, an additional abstraction level is introduced that hides both the particular cluster job ID and the type of computing environment. In our platform, a job running on, say, a computing cluster with the batch system has transparent access to the data received from jobs executed in other computing environments. Jobs running in different execution environments are displayed to the end user in a single list. From the point of view of the user, these have the same properties and control actions (e.g., submit for execution, get status, results).

The proposed software platform enables the adaptation of software for use in DCSs, providing an intuitive user interface for the entire spectrum of problems related to computational calculations performed by researchers. This technology allows an efficient use of computing resources, including user job management tasks, assigning the job to a computing environment, receiving and saving the computed data, providing a means of authorization and secure data exchange.

4 Example of adaptation: ASP FHI98md

To test the developed software platform, a problem-oriented web-based interface has been implemented for ASP FHI98md [10] and adapted to run in the Grid. This allows the user to directly perform the same procedure through a web browser as in the case of using the FHI98md console application installed locally on his PC.

The developed POI includes a server component, a web-based interface and the FHI98md ASP itself. Each is governed by its own configuration file, and can be located on a server that is geographically remote from the other components. The core of POI is the server component "abby", which generates the necessary files and control commands between the user web interface and the toolkit for constructing a sequence of Grid or other computing environment commands.

The web interface component "yyweb" provides functions for user authorization using x.509 certificates, validation of a set of job parameters, submission of the job to the server component and receipt of the necessary data. The REST architectural style and JSON data containers are used for data exchange.

The interaction of POI components is shown in Figure 2.

Abby is based on the Tornado framework used to create the so-called nonblocking web server. Interaction with other components is ensured by accepting HTTP/HTTPS-requests through the control socket. Request data are encoded in JSON format. Job objects, their status, file-related meta-information, etc., are stored in the SQLite3 database accessed using object relational mapping.
The server component is not involved in web form generation and interaction with the end user. That task is performed by the yyweb component, which is responsible for providing the end user with a web interface (including job configuration, submission for execution, and retrieving results). Yyweb collects the data required to generate job related files, and passes them to abby by calling certain API functions (Figure 3).

The developed POI allows creating a description of the computational tasks. It is composed of two elements: FHI98md configuration files and a job description file for the computing environment. Users are provided with a set of web forms and web pages including:

- problem specific data input form;
- computing environment job parameters input form;
- a page to monitor the progress of jobs, including the ability to cancel individual jobs;
- a page with links to access files containing job results.

The user inputs the parameters of the problem. The data are then passed along the following path:
1. From the browser on the user side, data are transmitted to the web application server on which yyweb is deployed. At this stage, the data entered by the user are validated to ensure they meet specific criteria (e.g., being within a certain range, correct dimensions, integrity of dependent values). An example of web form for data input is shown in Figure 4.

2. At the end of the verification process, the data are passed to abby. The obtained data are checked for completeness and integrity on the server side. Problem configuration files and computing job description files are then generated. If required, a supporting utility, fhi98start, is executed to create a problem configuration file on the basis of the generated files. The server component updates the database with information about the job, the files used and diagnostic information.

3. The resulting configuration files are transferred to the particular computing cluster that will carry out the calculations (Figure 5). Grid tools schedule the computing job, check the installed software compliance with the requirements of the job, verify the job description file, submit the job to the exact cluster batch system and transfer the job configuration files.

4. The job calculations start on one or more nodes of the computing cluster. If successful, a set of files containing the calculation results is generated; otherwise, the ASP or Job Scheduler generates an error report (Figure 6).
Figure 5: Web form to select a computing cluster that will perform the calculations (fragment).

Figure 6: Web form to access the computational job files for ASP FHI98md (fragment).

Note that using the Grid is beneficial for solving problems with relatively high computational costs. If the task is small, the overhead of setting it within the Grid cannot be justified. The calculation results are transferred to the user in reverse order.

1. All relevant files are returned to abby by service tools in the computing environment. In addition to the calculation results, the server also receives standard output and error output files.

2. The server updates the database with job information, marks the job as complete, and checks the availability of required files.

3. The user requests the server to obtain all the files via the web interface (Figure 7) or API.
At all stages, the user can access a web page with information about the execution state of the job. An authorized user can access these data from his personal workspace or via a unique URL.

5 Conlusion

In this study we have attempted to propose and develop the software platform that allows user interaction with application packages in the framework of distributed computing environments.

The principles of the proposed platform have been tested on the FHI98md ASP, which is a typical software package with complex logic in its configuration files and settings. The results suggest that the implemented platform is an effective solution.

Currently, the platform can be used for the adaptation of only those packages in which the input data are given in a text format, and does not support downloading of large files (e.g., image processing, etc.) via the web interface. We are planning to realize such an opportunity when developing future versions of the computing platforms.

Development of the platform includes the extension of functionality to work with additional input files downloaded by user. The ability will be implemented to upload files through the
web interface or by using other file transfer protocols from heterogeneous data sources (e.g., GridFTP).

The research results related to this software platform have already been implemented in several projects of the Far Eastern Branch of the Russian Academy of Sciences and third-party computing systems in Russia.

6 Acknowledgments

The study was supported by the Complex Fundamental Research Program of the Far Eastern Branch of the Russian Academy of Sciences ”Far East”.

References


