

Parametric Jobs – Facilitation of Instrument Elements Usage In Grid Applications

K. Bylec, S. Mueller, M. Pabiś, M. Wojtysiak, and P. Wolniewicz

Abstract The chapter discusses how to simplify integration of Grid applications in the Remote Instrumentation Infrastructure with usage of parametric extensions for JSDL. Capabilities of Parametric Sweep are presented spotlighting features most valuable in the frame of applying to Instrument Element enabled applications. Then a brief analysis of JSDL support in most common middlewares is followed by a real case study for one of the DORII project's applications. Starting from this application's use case two solutions in the field of parametric jobs are presented – g-Eclipse JSDL-Param library and Workflow Management System in DORII.

1 Introduction

Grid technologies are in constant development. Not only new tools emerge, but also the infrastructure elements are changing and new standards appearing. In the area of integrating new ideas into existing solutions the awareness of tools in the research field is significant. Combining together implementations from different levels can result in daily work improvements. This is a case for two relatively new Grid-related releases – Instrument Elements [1, 2], at the level of Grid infrastructure, and the Parameter Sweep [3] standard proposal, in the area of resources description languages.

With Instrument Elements the classical Grid infrastructure of computer and storage elements was enriched by the abstraction of real scientific devices. Providing Grid users with interfaces to instruments known from their daily work introduced new possibilities, but also the challenges. One of those challenges is to seamlessly integrate the Instrument Element into existing and newly created services in e-Infrastructures.

At the same time JSDL's [4] extension of Parameter Sweep was proposed to meet users' expectations for simplifying the description of Grid jobs that tend to become more complicated. Frequently their complexity emerges from a variety of

K. Bylec (✉)

Poznań Supercomputing and Networking Center, Poznań, Poland
e-mail: katarzyna.bylec@man.poznan.pl

possible input data sets as well as program execution options. Though the main logic of the application stays the same – the description of Grid jobs may vary significantly depending on different execution switches. Parameter Sweep allows to seize the abstract description of application's work in one part of the document, while spreading its execution to differently parameterized job instances with special extension, e.g. varying with input data.

In fact this is the data where Instrument Elements and Parameter Sweep meet. Instrument Elements seen as a virtualization of application's data sources may produce a vast amount of information to be processed by applications. Existing applications may not be prepared to handle different sets of data at the same time, especially when each set requires special processing. To make Instrument Elements of use without re-developing existing software or providing complex user interface solutions, we propose the adaptation of the Parameter Sweep in the field of Grid tools.

An introduction to the Instrument Element is briefly covered in Section 2. Then, starting from the above problem, the JSDL Parameter Sweep extension's possibilities are described in Section 3. Moving from user perspective to technical issues this work discusses the status of support for JSDL and its extensions in some Grid environments. The current state of middlewares' capability of handling the JSDL influenced the above problem's solution implementations – presented in Sections 4 and 5, while describing the g-Eclipse [5] platform and DORII's [6] Workflow Management System. The description of problems encountered by developers when introducing parametric jobs into their applications is followed by a presentation of the Java library for JSDL Parameter Sweep and a real case study for one of the DORII project applications.

2 Instrument Elements

Instrument Elements extend the classical Grid infrastructure of Computing and Storage Elements with the concept of virtualized and distributed scientific devices. They were introduced to the Grid with GRIDCC [7] and RINGrid [8] projects and since then – as the unified interface to program devices was provided – the tendency in Grid tools development started to change. The focus is moving from providing access to distributed computing and storage resources to enabling Grid applications with full access to scientific instruments – including the acquisition of experimental data, monitoring, managing, maintenance and troubleshooting.

Providing scientific communities with remote instrumentation results in improvement of collaborative work in distributed teams and increase in devices usage efficiency, as with remote access they are made available to a wider audience, which decreases the idle time of the experimental tools.

Nowadays an increasing interest in Remote Instrumentation Infrastructure is noticed worldwide – to mention only CIMA [9] in the USA (The Common Instrument Middleware Architecture, applying SOA to integrate sensors, seen as real time data sources, into the Grid), SpectoGrid [10] in Canada (exposing Nuclear Magnetic Resonance for remote access) and DORII [6] in Europe. DORII is

described in more detail in Section 5. of this work; here, we will mention only the devices this projects aims to enable for the e-Infrastructure:

- **Sensors of earthquake early warning system** – constantly register seismic movements to warn in case of increased activity (European Centre for Training and Research in Earthquake Engineering, Italy);
- **Floats and programmable Gliders** – oceanographic and coastal observation instruments to measure drifts with the currents at specific depths, temperature, salinity, oxygen, chlorophyll and turbidity profiles (Istituto Nazionale di Oceanografia e di Geofisica Sperimentale, Italy);
- **Digital cameras, pressure and temperature sensors** – registering images and conditions of coastal line and beaches for measuring users density and calculate the weather line (IH Cantabria – Universidad de Cantabria, Spain);
- **X-Ray Diffraction detector** – data acquisition detectors for Diffraction Beamline supporting scientists in controlling if the data acquisition process is meeting quality requirements and for on-line data analysis (Sincrotrone Trieste ScpA, Italy);
- **Optical sensors for monitoring inland waters** – provide information in near real time about the water quality status (Ecohydros SL, Spain).

The above small subset from the category of Instrument Elements illustrates the main problem in the field of creating a homogenous interface of Remote Instrumentation Infrastructure – this is the heterogeneity of the domain. Instrument Elements range from all-time-up sensors to devices run on demand, which may require advanced reservation and accounting algorithms. Aside of the access scenarios, devices vary with network traffic they rise – from constant and low to irregular and bulk data transfers. This variety creates one of the reasons for bringing to life the OGF Remote Instrumentation Services in Grid Environment Research Group [11] (RISGE-RG), which gathers representatives from different fields and projects to collect existing solutions in accessing remote devices and aims at seizing the abstract approach for Remote Instrumentation. Its work will result in defining the foundation of an OGF Working Group, which will manage standardization in the field of Instrument Elements.

Though being relatively new in the Grid science, Instrument Elements acquire increasing interest. The scientific community seems to notice the fact that with Remote Instrumentation Infrastructure they may enable their work with devices for high performance and gain access to remote and distributed tools. Having this in mind we should focus on tools and technologies that would facilitate the usage of Instrument Elements in users' daily work. Such tools would be especially those with standards behind them – e.g. JDSL.

3 JSDL

Grid accessibility can be expressed with the power and capability of basic tools to accomplish standard Grid operations including job submission and data operations. Among them a well designed job description language is one of the most important

Grid features. In this section an OGF standard within this domain will be presented: the Job Submission Description Language (JSDL). A brief description of the general JSDL capabilities (3.1) is followed by an attempt to characterize its popularity within Grid middleware and middleware access tools (3.2). The parameter sweep feature is presented in more detail – discussing its power to express different parametric use cases (3.3). The section is completed with a description of the middleware capacity to handle parameter sweep (3.4).

3.1 JSDL Non-Functional Features

The Job Submission Description Language (JSDL) [4] is an XML-based computational¹ resource description language developed by OGF JSDL-WG [12]. The project was launched in Sept. 2003, and now it is publicly available in version 1.0 of the specification.

It is worth noticing that in JSDL-WG's goal, which is to specify an abstract standard of job description language independent of underlying middleware, JSDL is thought as a superset of other popular job description languages such as JDL or RSL, but also those e.g. based on Web Services invocations.

Because of this concept its structure is composed of the general purpose description part and domain or technology specific extensions.² Among those extensions are [13]:

- **POSIX Application** – defines which executable host destinations are POSIX standard compatible machines;
- **HPC Profile Application** – describes executables to be run as operating system processes;
- **SPMD Application** – Single Program Multiple Data extension for describing parallel applications with single executable;
- **Parameter Sweep** [3] – abstract description of jobs collection varying with some parameters as a single job, the *Proposed Recommendation* by OGF.³

3.2 JSDL Popularity

Because of its general and abstract way of describing jobs, JSDL is a very powerful language. Even if some of the available extensions are lacking desired functionalities JSDL's users may provide their own extensions – not only at the level

¹Though it is named *computational* on the JSDL-WG web page it can be also used for data transfer operations alone.

²The extensible architecture is becoming a popular solution in resource description; it is also introduced in the Job Description (JD) Document by Globus Toolkit version 4.2.1. The difference is that Globus' GRAM4 was designed to support a single extension point (for Local Resource Manager) which is one of the reasons this middleware is lacking support for JSDL. See Table 1. for details.

³Announced 12.05.2009

Table 1 Support for JSDL in Grid middlewares

Middleware	Default JOB Description Language	JSDL Support	Details of JSDL support
gLite 3.2	Job Description Language (JDL)	Yes	As a patch to WMS UI 3.3, simple jobs only, plan to add bulk submission [14]. In older version it is possible to convert JSDL to JDL [15] with XSLT provided by OMII-Europe [16].
Globus Toolkit 2	Resource Specification Language (RSL)	No	N/A
Globus Toolkit 4	Job Description Document (JDD aka RSL 4)	Yes	Since version 4.0.5 GT is provided with GridWay Metascheduler that takes care of JSDL submission [17]. It also has support for JSDL HPC profile.
UNICORE 6	JSDL in server services (JSON in the UCC)		Job descriptions in JSON format are handled by the server side flexible execution backend (XNJS).

of job type but also in every existing XML element and attribute. Another important fact about JSDL is it has OGF support and right now should be considered a standard. Knowing all this one would expect JSDL being popular in Grid application domains, which is not the case. The only reasonable explanation for this state are historical reasons. Many of the Grid middlewares and tools developed for them have started before JSDL has confirmed its position. Some examples of introducing JSDL to Grid projects are presented in Table 1.

It is remarkable that none the most popular Grid middlewares made JSDL their default job description language. Support for it is provided in form of patches or additional tools. This solution is justified in cases when backward compatibility is a priority, but some of the projects (e.g. Globus Toolkit 4) introduced completely changed job descriptions instead of making a standard one their main format for describing Grid jobs. Therefore it is understandable that scientists tend to use old tools, even if usage of them means that the efficiency of work will not increase. It is not in the scope of this chapter to analyse the popularity of the JSDL language and reasons for it, nevertheless a quick conclusion here would be that even within the Grid domain (not so popular itself among other scientific fields) ideas and their releases lack good dissemination. This leads to sticking to old implementations and inhibition of spreading of new technologies. We are facing a vicious circle in case of JSDL and that is the reason why all efforts to support new ideas should be even more promoted, mainly by pointing fields in which their application may result in increase of efficiency. This is the case for introducing a Parametric Sweep extension to Remote Instrumentation Infrastructure.

3.3 Parameter Sweep Extension

The intuitive understanding of parameterizing a job is correct when it comes to Parameter Sweep [3]. It would be defined as *describing as a single job a set of many jobs varying among them with some detail*. This detail may be, e.g., the option of an executable or the name of an input file.

The above idea is not new in the Grid environment. gLite's JDL also supports parameters [18] (to be more precise – one parameter), but the difference here is significant; that is why we will provide a short description of Parameter Sweep capabilities, which are not limited to number of parameters, functions describing the values to be substituted or the ordering of parameters sweeping. All those aspects are covered in the following subsections.

3.3.1 Target of Substitution

The parameter element specifies which JSDL element should be substituted when resolving JSDL with parameters. The user can decide to sweep:

- **Any XML element or attribute of JSDL** – a value of XML element or attribute can be referenced using the XPath language [19]. Moreover XPath's substring function can be used to point only to part of the referenced string value. This functionality is covered with DocumentNode element.
- **A token inside a file external to JSDL** – in the same way as values inside a JSDL element are substituted, external text files can be modified. This is achieved through the FileSweep element.

The difference in behavior of FileSweep and DocumentNode is that the latter accesses values of XML entities pointed by name whereas FileSweep's input is a token, which will be substituted. In other words – after applying sweep to DocumentNode the user will still have the names of referenced XML entities with changed values and in case of FileSweep the token will no longer be present in the file, as it was replaced with the value given.

3.3.2 Values to be Substituted

Values to be swept for parameters are defined with the Function element. This is an abstract element and therefore can be seen as an extension point of the Parameter Sweep specification – the user can define a structure to meet his substitution requirements. By default Parameter Sweep provides three kinds of Functions (which don't have to be "function" in mathematical sense). Those are:

- **Array of values** – ordered set of unrelated data of the same type;
- **Loop** – loops are defined with starting, terminating and step values; it is also possible to exclude some of the values from the loop's range. The specification provides two default loop functions: LoopInteger and LoopDouble for integer and double values, respectively.

3.3.3 Order of Substitution

In case of many parameters definitions it is possible to steer the order of values substitution. This functionality is captured by the Assignment element, which couples the parameter with values to be swept for it. Because it is possible to nest Assignment elements as a side effect the user gains the ability of defining the dependencies between different sweeps as follows:

- **Sweep at the same time** – at each sweep of values one value is substituted for each parameter defined within JSDL. Parameters may be assigned to different functions and therefore may take different values at the same time. It is also possible to assign one function to more than one parameter. The number of resulting jobs is equal to the values set cardinality.
- **Independent sweep** – whole substitution for one parameter is done at one time while the rest of parameters take their default value. The number of resulting jobs is equal to the sum of all values sets cardinalities.
- **Nested sweep** – for one sweep of value of external parameter the whole substitution (of all values) is done for the internal one. The number of resulting jobs is equal to the multiplicity of values sets cardinalities for the nested elements.

3.4 Parametric Jobs at Middleware Level

We must stress the difference between job description and submission of job. The fact that the user is able to specify a set of jobs with one description is not equal to submitting only one job. It is up to the middleware implementation if after submission the parametric job will be seen as a single collection of jobs or as many independent Grid jobs. The difference concerns e.g. the execution environment settings or sharing of input/output files. The concept of a collection of jobs [18] and shared sandbox is known from gLite and JDL language. In this case JDL provides the ability to decide on a shared sandbox and gLite middleware is able to fulfill this requirement. In case of Parameter Sweep for now there's no support for submission of a parametric job as a collection of jobs. At the level of JSDL it would be possible with a simple attribute extension, but we are still missing support from middlewares.

At the time of writing this chapter none of the existing Grid middlewares could handle the Parameter Sweep extension. Nevertheless in the following sections we will try to show that regardless of this fact using parametric jobs is reasonable. Introducing Parameter Sweep even only at the level of Grid clients results in simplification of user interfaces. It leads to abstraction of the job to be done and therefore allows to encapsulate the whole work delegated to the Grid as a workflow.

4 G-Eclipse Implementation

g-Eclipse [5] is project founded by the European Commission under the 6th Framework Programme (it lasted from July 2006 to December 2008) as well as the Eclipse Foundation [20] project. It builds on top of the Eclipse platform and

Table 2 g-Eclipse's middlewares support

Middleware	Available functionalities
GLite	Authentication and authorization – VOMS proxy extension; job management – editing, submission, monitoring (JDL/JSDL); data management (gsiFTP, LFC, RSM); GLUE information system; remote application debugging and deployment; workflow submission.
GRIA	Job management – editing, submission, monitoring (JSDL); data management (GRIA file store).
Globus Toolkit 2	Authentication and authorization – Globus proxy; job management – editing, submission, monitoring (RSL); data management (gsiFTP); gLogin.
Globus Toolkit 4	Support for GT 4.2 (not compatible with GT 4.0) Job management – editing, submission, monitoring (JDD aka RSL 4); information system (MDS)

provides a middleware independent architecture of workbench tools for accessing Grid resources. It is targeted at three users groups: Grid Users, Grid Application Developers and Grid Operators. g-Eclipse's exemplary middleware target was gLite, but support for other e-Infrastructure was also implemented – details are provided in Table 2.

Though designed and implemented as a graphical user tool, g-Eclipse can be also utilised as a Grid access library, especially taking into account work done within the DORII project (see Section 5.1 *WfMS and Common Library*). This approach is discussed in the following section, but first the original g-Eclipse solution is presented.

4.1 g-Eclipse JSDL Plug-Ins

g-Eclipse extends the Eclipse Platform's plug-ins architecture. It is composed out of layered plug-ins: at the bottom there are Eclipse's standard UI and resources management as well as extensions mechanism, on top of them the abstraction layer for Grid functionalities is build which is then extended – if needed – with middleware specific [21].

JSDL is introduced at the middle level – as the standard job description language for middleware abstraction. Still it is possible to submit the job description in middleware specific language but the recommended and – to some point – forced by architecture solution is to operate on the JSDL file and then transform it to specific language just before submission to the middleware.

g-Eclipse simplifies this recommended job management flow. It provides the JSDL Java model for developers and multipage graphical editor which hides the JSDL XML's complexity from the user. Supported JSDL extensions are POSIX and Parameter Sweep.

Because none of the supported middlewares handles JSDL Parameter Sweep, the parameterized JSDL files (compare Section 3) have to be preprocessed before submission. Preprocessing involves:

1. **resolving of parameters** – sweeping all values and generating resulting JSDL files without Parameter Sweep extension's elements;
2. **submission of each JSDL file** generated in the first step as an independent Grid job.

In the GUI – in Project view as well as in Jobs monitoring view – the parametric is represented as a single job with resulting swept files as children. Before the submission user is given the possibility to preview the values' array that will be used for swept job descriptions. This can be done in JSDL editor and won't affect the parametric JSDL file.

4.2 Java Library for Parameter Sweep

g-Eclipse is build on top of the Eclipse Platform and therefore depends on its architecture as well as on some low level model code e.g. for resources management. This is the reason why, though g-Eclipse's JSDL functionality is encapsulated within 3 plug-ins, it cannot be used outside of the Eclipse Platform. It would be a shame if this code – providing wide coverage of POSIX extension and one of the few existing implementations of Parameter Sweep – could not be reused by other projects. JSDL plug-ins can be easily integrated into RPC applications [22], but it is not always the case that an external project can be converted into Eclipse application. That is why we extracted parts responsible for Parameter Sweep handling. They are available in form of Eclipse independent Java library called Param-JSDL [23] under Eclipse Public Licence [24].

This library is independent of the g-Eclipse JSDL model – it operates on string elements. The entry point to param-jsdl.jar is the `IParametricJsdlGenerator` interface and its single method generator, which in turn takes `IParametricJsdlHandler` instance as an argument. `IParametricJsdlHandler` is a listener implementation for events that occur during processing of parametrized JSDL. Those are: start of generation, generation of new JSDL, cancel and finish, which developers can customize to suit their requirements.

The implementation in the background, to which instances of `IParametricJsdlHandler` register for generation events information, uses DOM XML representation and XPath language for traversing and modifying the JSDL structure. XML processing is done in memory – no I/O operations are involved – which results in efficiency increase (Fig. 1).

The speed of generation depends on the structure of JSDL and parameters number. It doesn't depend on parameters values. Results of tests performed on a middle class developer's PC computer (Intel ® Core™ 2 CPU, T5500 @ 1.6 GHz, 2 GB RAM, Windows Vista™ Home Basic 32-bits, Java™ SE Runtime Environment, build 1.6.0_13-b03)) are visible in Figs. 2, 3, and 4. As a basic reference data set a JSDL with executable definition and one data staging was used, starting from one parameter with 3 values, whose number was increasing up to 15. For each JSDL 1500 substitutions were performed.

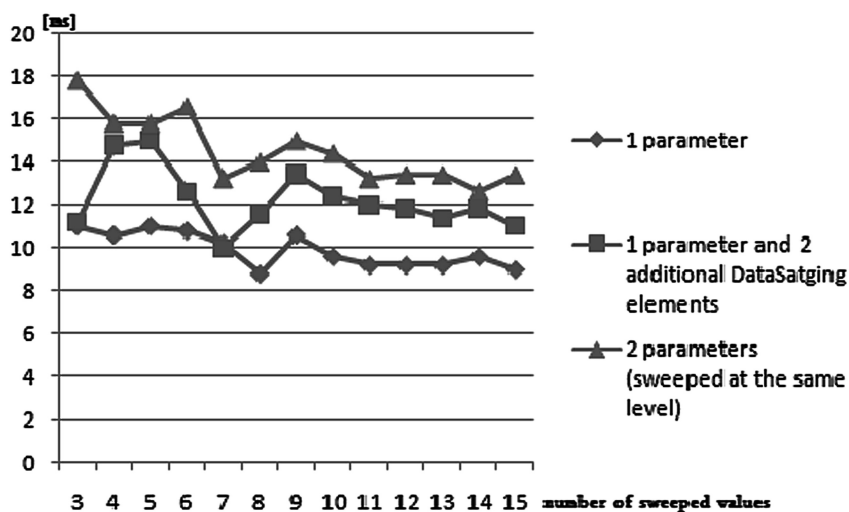


Fig. 1 Performance test results of JSDL-Param (own elaboration)

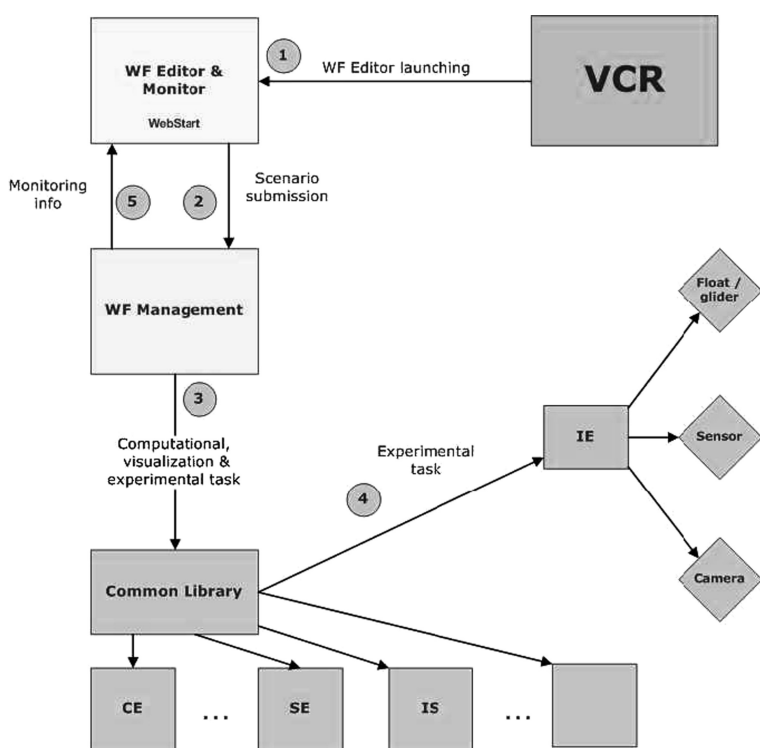


Fig. 2 Scheme of WfMS in DORII

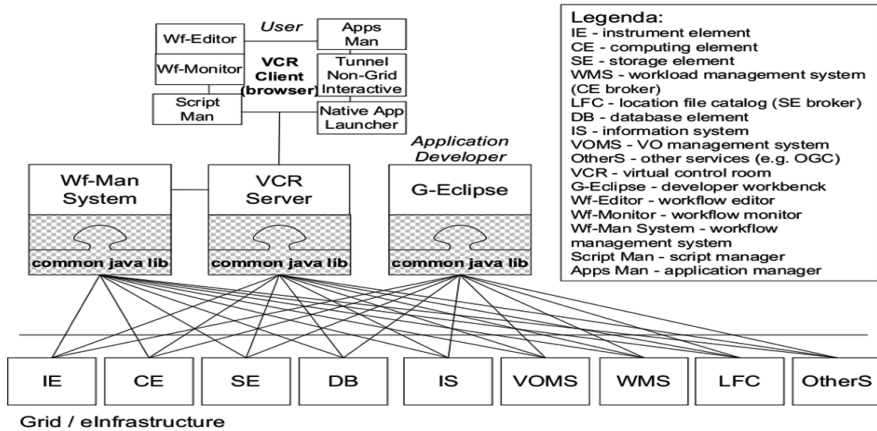


Fig. 3 DORII architecture

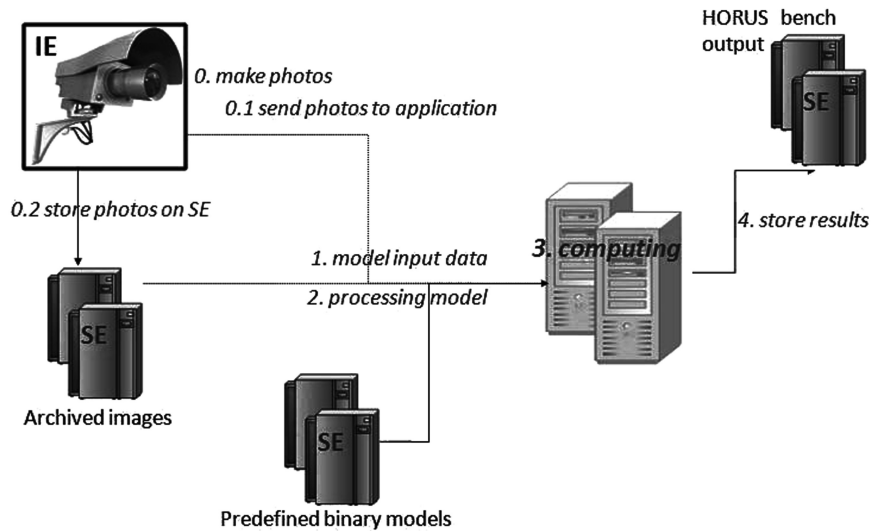


Fig. 4 HORUS_bench workflow

5 DORII Implementation

Deployment Of Remote Instrumentation Infrastructure (DORII) [6, 25] aims to introduce the Grid extended by Instrument Elements to scientific communities such as Earthquake community (with various sensor networks), Environmental science community, Experimental science community (with synchrotron and free electron lasers). The project goal is to integrate existing Instrument Elements with the Grid

infrastructure and develop applications handling them as well as provide high-end tools with a GUI to simplify maintenance and access to the deployed structure.

When first versions of Instrument Elements had been successfully installed users were able to start using high-level tools, among them – the VCR developed initially in GridCC [7] and the Workflow Management System, first introduced by the VLab project [26], whose idea was deployed with success e.g. in the EXPReS project [27, 28].

5.1 *WfMS and Common Library*

The Workflow Management System (WfMS) was designed to support users in running, managing and monitoring Grid applications instances, defined as a sequence of Grid jobs, i.e. workflows. In the DORII project workflows involve Instrument Elements, that may take place in the every part of the chain – being source of data for other workflows component as well as the data destination.

WfMS is composed out of two parts – GUI Workflow Editor and server side Workflow Manager. The Editor may be launched independently (as a Java Web Start application) or from within a VCR instance. It servers a workbench with predefined workflow instances that the user customizes before its submission to the Grid – specifying data and steering the flow between components, describing the experiment parameters and conditions. The workflow’s definition may contain components representing operations on Instrument Elements, Grid job execution, storage access or prompting for user input if required. Grid components behind each of the workflow’s block are restricted by the Virtual Organization (VO) of the user.

The workflow is started by the user from the Editor and then the steering is passed to the server side component. This is the Workflow Manager that is responsible for determining the order of the workflow’s parts to be executed as well as for transforming them to concrete Grid operations (e.g. generating JSDLs, executing transitions or operations on Instrument Elements), for monitoring execution and managing data transfers. The server component of WfMS acts as a central management point of workflow execution and as a Grid access interface. For the latter it uses the Common Library component also developed within the DORII project. Details of the design are covered by Fig. 2.

The Common Library is a Java library designed to be a common access and functionality layer between integrated e-Infrastructure components. The idea that resulted in the Common Library creation was the observation that libraries for accessing the Grid infrastructure are distributed over many different projects with different goals and objectives, which means they may vary with technology (e.g. client or server side, web services based, etc.), programming language (Java or C) or may not be complete. The Common Library component exposes Grid access interfaces for services such as MyProxy and Instrument Element taken from EGEE and DORII projects, respectively, as well as Information System (BDII), WMS, and data management (gsiFTP, LFC, SRM) – extracted from g-Eclipse.

The relation between components described above is captured in Fig. 3 representing the DORII architecture.

5.2 *HORUS_bench Application*

HORUS_bench is one of the applications defined by the DORII project to be integrated into the Remote Instrumentation Infrastructure. It is developed by Instituto de Hidráulica Ambiental, Universidad de Cantabria, Spain and concerns the environmental data analysis.

HORUS_bench involves digital cameras, temperature and pressure sensors as Instrument Elements. Data gathered by them is used to monitor and simulate conditions of the beaches and coastal line of sea and rivers – especially calculating the water line, user density on the beaches, etc. Because of the distributed geographical nature of those devices as well as amounts of data they produce, scientists are lively interested into introducing HORUS_bench into the e-Infrastructure.

The use case is (see also Fig. 4):

- The gather input data from Instrument Elements – e.g. cameras taking pictures of beaches – they will come as an input stream to the application or be stored on SE and read from it;
- From the list of available image processing algorithms the user chooses the set that will be applied to input data. The chosen set of algorithms will constitute a binary file of the application;
- Application's input data and algorithms (in form of binary file) are taken by the HORUS_bench script run on the CE – the set of algorithms is applied to each of the images from input data gathered by Instrument Elements;
- Each file produced as a result of HORUS_bench processing is transferred as an output to a remote SE location.

5.3 *Applying Parametric Jobs for HORUS_bench*

Assuming that the HORUS_bench script takes one input file at once and this file has to be available on the script's path at the execution time, we propose the approach of submitting many HORUS_bench jobs, one for each file from the input set. Those jobs will be run in parallel, writing result files into the same remote location. This means that each job will have the same set of processing algorithms and the same executable. The only changing parameter will be the input file name (as the location stays the same).

Starting from the point where the user launches the HORUS_bench workflow from the Editor – the Workflow Manager transforms the information provided (input images location and processing algorithms) into a Parametric JSDL file. The simplified job description is presented in Fig. 5.

The solution described at the beginning of this subsection is reflected in JSDL design – parameterization is applied to job's data stagings (described in XPath language as `/*/jsdl:DataStaging[3]/jsdl:Source/jsdl:URI`), but only to the file name part of JSDL's element (using XPath `substring(//*/jsdl:DataStaging[3]/jsdl:Source/jsdl:URI, 54, 1)` function).

```

<?xml version="1.0" encoding="UTF-8"?>
<jSDL:JobDefinition>
  <jSDL:JobDescription>
    <!-- ... -->
    <jSDL:Application>
      <jSDL:ApplicationName>HORUS_bench</jSDL:ApplicationName>
      <jSDL-posix:POSIXApplication>
        <jSDL-posix:Executable>horus.sh</jSDL-posix:Executable>
      </jSDL-posix:POSIXApplication>
    </jSDL:Application>
    <jSDL:DataStaging>
      <jSDL:FileName>horus.sh</jSDL:FileName>
      <!-- ... -->
    </jSDL:DataStaging>
    <jSDL:DataStaging>
      <jSDL:FileName>algorithm_model_bin</jSDL:FileName>
      <!-- ... -->
    </jSDL:DataStaging>
    <jSDL:DataStaging>
      <jSDL:FileName>file1.png</jSDL:FileName>
      <jSDL:Source>
        <jSDL:URI>gsiftp://path/on/SE/to/Instrument/Element/output/file1.png</jSDL:URI>
      </jSDL:Source>
    </jSDL:DataStaging>
  </jSDL:JobDescription>
  <sweep:Sweep>
    <sweep:Assignment>
      <sweep:Match>substring(/**//jSDL:DataStaging[3]/jSDL:Source/jSDL:URI, 54, 1)</sweep:Match>
      <sweep:Match>substring(/**//jSDL:DataStaging[3]/jSDL:FileName, 5, 1)</sweep:Match>
      <sweepfunc:LoopInteger sweepfunc:end="99" sweepfunc:start="0" sweepfunc:step="1"/>
    </sweep:Assignment>
  </sweep:Sweep>
</jSDL:JobDefinition>

```

Fig. 5 Simplified parametric JSDL for HORUS_bench

It is worth noticing that in case of applying parameterization to data stagings elements two sweeps have to be defined. One for the resource path (as stated in the previous paragraph) and another for the name of the staged file. This may be seen as a disadvantage for users of Parameter Sweep, nevertheless before turning this into an entry point for discussion on user friendliness of the specification, we should be aware that Parameter Sweep is a low level language. It should be the users' tools responsibility to handle and hide such a situation from the user. Moreover coupling between data staging's path and its name is not always the case. The JSDL specification states that the FileName element defines a name used for file in sandbox, while the path points to a resource. In other words – during transfer of data staging to a sandbox its name may be changed.

The latter use case would be of use if we proposed a different approach to HORUS_bench parameterization. In the solution described above (sweep of input file's path and name) the application's script just looks for an image file on its execution path. We may imagine that this script was developed to read always a file of the same name (the input file name is hardcoded in the script). In that case two solutions are possible:

- To parameterize only the input file path in JSDL (not the FileName element which will take the hardcoded value from the script);
- To parameterize the input file path and name in JSDL and to apply the FileToken sweep to the HORUS_bench script).

The former is not much different from the solution chosen in WfMS, so we will skip its description. The latter would be only possible if the script is a local file, as it should be available at the moment of Parameter Sweep processing of JSDL. Having the script as a Grid storage location means that it is accessible only at the moment of execution of the job in the Grid sandbox, where neither the Parameter Sweep reference within JSDL nor the token definition inside the script would be properly interpreted.

When the WfMS has finished generation of JSDL from Fig. 4 the job description is submitted to the Grid using the Common Library's WMS interfaces. The way of handling JSDL by Common Library is based on g-Eclipse's model implying that the JSDL with Parameter Sweep is validated and then parsed by JSDL-Param library, which results in many non-parameterized JSDL descriptions, separately maintained by Common-Library in the process of Grid submission and monitoring.

5.4 Applying Parameterization at Different Levels

Details of the solution described above are distributed into two levels – implementation involves functionalities of the WfMS and Common Library components. In fact the lowest level is missing for the solution to be considered complete – the middleware support.

In gLite middleware while submitting a single job before the to a WMS the job is given its own sandbox. Data staging files are transferred to sandbox before the application is launched and outputs transferred to target destination after it has finished. Running HORUS_bench for a single input file out of GB of input results in heavy traffic load, not to mention the overload of WMS and job queues. That is why the presented solution is only a partial one for the described problem. It reduces the complexity of workflow at the level of user interface – it allows to hide details of the undelaying implementation and present the user only the main application's idea which, in this case, is bulk, not individual file processing. The server side workflow maintenance is also simplified – in the Computing Element block the steering float does not have to fork for many single jobs, as this is handled by the underlying Common Library's JSDL model and Param-JSDL library described in previous sections. Unfortunately, the lack middleware capability for handling Parameter Sweep does not allow for completing the process of facilitation of the Instrument Element adaptation into HORUS_bench. Nevertheless – in comparison to the non-parametric description of HORUS_bench the gain is significant, measured mainly with the possibility of Grid enabling of the application.

6 Conclusions

Enabling existing and newly created services for a Remote Instrumentation Interface gains increasing interest among scientific fields. The expectation of efficiency improvement can be met by combining Instrument Elements usage with JSDL's

extension for parametric jobs – Parameter Sweep. First attempts to achieve this goal were made within the DORII project and its Workflow Management System solution for a sequence of Grid operations (including remote devices access as well as jobs submission). This was possible with the outcome of another project – g-Eclipse, whose early experience in Parameter Sweep could be ported to DORII.

Though the Parameter Sweep is of much help for enabling applications for Instrument Elements still the solution is not complete, as the middlewares lack the capability of JSDL and Sweep Parameter handling.

References

1. E. Frizzierol, M. Gulminil, F. Lelli, G. Maron, A. Oh, S. Orlando, A. Petrucci, S. Squizzato, and S. Traldil. Instrument Element: a new Grid component that enables the control of remote instrumentation. Workshop on Scientific Instruments and Sensors on the Grid, Trieste-Italy, 23–28 April, 2007
2. M. Plóciennik. RISGE-RG Collection of use cases, OGF Document Series web site, document GFD.168. <http://ogf.org/documents/GFD.168.pdf>, last accessed September 2010
3. JSDL Parameter Sweep Job Extension; <http://www.ogf.org/documents/GFD.149.pdf>, last accessed August 2009
4. Job Submission Description Language (JSDL) Specification, Version 1.0; <http://www.ogf.org/documents/GDF.136/pdf>, last accessed August 2009
5. g-Eclipse, project website, <http://www.geclipse.eu>, last accessed August 2009
6. Deployment of Remote Instrumentation Infrastructure (DORII), project website, <http://www.dorii.eu>, last accessed August 2009
7. GRIDCC, project website, <http://www.gridcc.org>, last accessed August 2009
8. RINGrid, project website, <http://www.ringrid.eu>, last accessed August 2009
9. T. Devadithya, K. Chiu, K. Huffman, D.F. McMullen. The common instrument middleware architecture: Overview of goals and implementation, http://cs.indiana.edu/~tdevadit/pubs/cima_isog05.pdf, last accessed August 2009
10. SpectroGrid, project webpage <https://spectrogrid2.nrc.ca>, last accessed August 2009
11. Remote Instrumentation Services In Grid Environment – RG (RISGE-RG), OGF Research Group webpage, <https://forge.gridforum.org/sf/projects/risge-rg>, last accessed August 2009
12. Job Submission Description Language WG, OGF Working Group webpage, <https://forge.gridforum.org/sf/sfmain/do/viewProject/projects.jsdl-wg>, last accessed August 2009
13. JSDL-WG Published Documents: JSDL HPC Profile Application Extension Version 1.0 and JSDL SPMD Application Extension Version 1.0, https://forge.gridforum.org/sf/docman/do/listDocuments/projects.jsdl-wg/docman.root.published_docs, last accessed August 2009
14. Gruppo GRID Datamat EGEE Wiki Pages, Patch #3040, <http://trinity.datamat.it/projects/EGEE/wiki/wiki.php>, last accessed August 2009
15. OMII-EU JRA1/Job Submission Task Wiki Pages, JSDL to JDL converter <http://grid.pd.infn.it/omii/jsdl2jdl>, last accessed August 2009
16. OMII-Europe, project webpage, <http://www.omii-europe.org/>, last accessed August 2009
17. Globus Alliance Community Webpage, GT 4.0.5 Incremental Release Notes, <http://www.globus.org/toolkit/releasenotes/4.0.5/>, last accessed August 2009
18. Gruppo GRID Datamat EGEE Wiki Pages, JDL Types, <http://trinity.datamat.it/projects/EGEE/wiki/wiki.php?n=JDL.JDLTypes>, last accessed August 2009
19. XML Path Language (XPath), Version 1.0, W3C Recommendation, <http://www.w3.org/TR/xpath>, last accessed August 2009

20. g-Eclipse: Tools for Grid and Cloud Computing, project webpage on Eclipse Foundation portal, <http://eclipse.org/geclipse/>, last accessed August 2009
21. g-Eclipse Consortium, g-Eclipse Final Architecture, <http://www.geclipse.org/fileadmin/Documents/Deliverables/D1.8.pdf>, last accessed August 2009
22. FZK's Savannah CVS source code repository, g-Eclipse example RPC applications – Gaussian, http://savannah.fzk.de/cgi-bin/viewcvs.cgi/fzk/geclipse/geclipse/demo_apps/eu.geclipse.gaussian.jmol/, last accessed August 2009
23. FZK's Savannah CVS source code repository, JSDL-Para library, http://savannah.fzk.de/cgi-bin/viewcvs.cgi/fzk/geclipse/geclipse/demo_apps/JSDL_Param/, last accessed August 2009
24. Eclipse Foundation, Eclipse Public License, version 1.0, <http://www.eclipse.org/legal/epl-v10.html>, last accessed August 2009
25. A. Chepstov, R. Keller, R. Pugliese, M. Prica, A. Del Linz, M. Plociennik, M. Lawenda, and N. Meyer. Towards deployment of remote instrumentation e-infrastructure (in the Frame of the DORII Project). *Computational Methods in Science and Technology*, 15(1), 65–74, 2009
26. Virtual Laboratory (VLAB) project webpage, <http://vlab.psnc.pl/>
27. EXPRéS project website, <http://www.expres-eu.org/>
28. M. Okon, D. Stokłosa, R. Oerlemans, H. J. van Langevelde, D. Kaliszan, M. Lawenda, T. Rajtar, N. Meyer, and M. Stroinski. Grid integration of future arrays of broadband radio-telescopes moving towards e-VLBI. *Grid enabled remote instrumentation*. Springer, p. 571, 2008

Remote Instrumentation Services on the
e-Infrastructure

Applications and Tools

Davoli, F.; Meyer, N.; Pugliese, R.; Zappatore, S. (Eds.)

2011, XVII, 325 p., Hardcover

ISBN: 978-1-4419-5573-9