



# PROCEDURES FOR INTEGRATION OF APPLICATIONS WITH MIGRATING DESKTOP

---

Document Filename: **BG-DNA3.2-v1.5-PSNC-MDIntegrationProcedures**

Activity: **NA3**

Partner(s): **PSNC**

Lead Partner: **VU**

Document classification: **PUBLIC**

---

**Abstract:** The BalticGRID's deliverable DNA3.2 presents the results of analysis based on collected BalticGrid applications description and requirements and the Migrating Desktop (MD) platform technical specification. The document aims to provide a technical description of procedures of integrating the Migrating Desktop with applications. The technical analysis leads to a classification of applications regarding to features required by the GUI side for grid operations (job preparing, submitting, and processing the results). Possibilities and technical aspects of integration are presented and described for each feature. Based on this analysis the application developers will be able to make decisions concerning the integration. This document also gives important feedback to the Migrating Desktop developers concerning new requirements not yet supported.

## Document review and moderation

	Name	Partner	Date	Signature
Released for moderation to				
Approved for delivery by				





### Document Log

Version	Date	Summary of changes	Author
0.1	03/4/2006	Draft version	Marcin Płóciennik
0.4	10/4/2006	Draft version	Bartosz Palak, Marcin Płóciennik
0.5	14/4/2006	Draft version. Added integration procedures, appendix	Bartosz Palak, Marcin Płóciennik
1.0	21/4/2006	Added executive summary, improvement in English language, Changes in requirement analysis	Marcin Płóciennik, Paweł Wolniewicz, Mirosław Kupczyk, Norbert Meyer
1.1	8/5/2006	Added OCM-G part	Tomasz Szepieniec
1.2	11/5/2006	Improvements and corrections.	Marcin Płóciennik, Bartek Palak,
1.4	16/5/2006	Improvements and corrections.	Marcin Płóciennik, Bartek Palak, Algimantas Juozapavicius, Daiva Kaukeniene
1.5	13/06/06	Corrections from internal review	Bartek Palak



## CONTENTS

<b>1. INTRODUCTION.....</b>	<b>5</b>
1.1. PURPOSE OF THE DOCUMENT .....	5
1.2. ABBREVIATIONS .....	5
<b>2. EXECUTIVE SUMMARY .....</b>	<b>6</b>
<b>3. BALTICGRID APPLICATION REQUIREMENTS ANALISYS.....</b>	<b>7</b>
3.1. BALTICGRID APPLICATIONS .....	7
3.1.1. Analyzed applications .....	7
3.1.2. Other applications .....	8
3.2. APPLICATION REQUIREMENTS .....	8
3.3. APPLICATION SPECIFIC REMARKS.....	17
3.4. APPLICATION REQUIREMENTS REVIEW .....	19
3.4.1. Application input.....	19
3.4.2. Preparing the input .....	20
3.4.3. Job submission .....	20
3.4.4. Parallelisation.....	20
3.4.5. Running modes.....	20
3.4.6. Multiply jobs and their interdependencies.....	20
3.4.7. Output format.....	21
3.4.8. Partial results.....	21
3.4.9. Visualisation.....	21
3.4.10. Processing results .....	22
3.5. SECURITY REQUIREMENTS.....	22
<b>4. INTEGRATION WITH THE MIGRATING DESKTOP .....</b>	<b>23</b>
4.1. MIGRATING DESKTOP OVERVIEW .....	23
4.2. DESCRIPTION OF THE MIGRATING DESKTOP FUNCTIONALITY WITH RESPECT TO THE BALTIC GRID APPLICATION REQUIREMENTS .....	24
4.2.1. Application input.....	24
4.2.2. PREPARING INPUT .....	25
4.2.3. Job submission .....	27
4.2.4. Running modes.....	27
4.2.5. Multiply jobs and their interdependencies.....	28
4.2.6. Partial results.....	29
4.2.7. Processing the output .....	30
4.2.8. Visualisation.....	30
4.2.9. Security requirements.....	30
4.3. COMMENTS TO “SPECIFIC APPLICATION REMARKS” .....	30
<b>5. INTEGRATION PROCEDURES.....</b>	<b>31</b>
5.1. GENERAL IDEA OF PLUG-INS .....	31
5.2. OSGI.....	31
5.2.1. OSGi bundles .....	32
5.2.2. OSGi services.....	32
5.3. PLUG-INS ARCHITECTURE .....	32
5.3.1. Plug-in, toolkit and container .....	32
5.3.2. Plug-in components interdependencies .....	33
5.4. POINTS OF INTEGRATION/TYPES OF PLUGINS .....	33
5.4.1. Job Input Plug-in (IP1& IP2) .....	33
5.4.2. Job Input Plug-in API.....	34
5.4.3. Job Input Plug-in Toolkit API.....	35
5.4.4. Parameters passed to job viewer plug-in .....	38
5.4.5. XML schema description of job specific input parameters .....	39
5.4.6. Job Process Plug-in (IP3) .....	44
5.4.7. Job Process Plug-in API .....	44
5.4.8. Job Process Plug-in Toolkit API .....	45



**PROCEDURES OF INTEGRATING APPLICATIONS WITH  
THE MIGRATING DESKTOP**

---

5.4.9.	File Viewer Plug-in (IP4).....	48
5.4.10.	File Viewer Plug-in API.....	48
5.4.11.	File Viewer Toolkit API.....	49
5.4.12.	Job Viewer Plug-in (IP5 & IP6).....	51
5.4.13.	Job Viewer Plug-in API.....	51
5.4.14.	Job Viewer Toolkit API.....	52
5.4.15.	Tool Plug-in.....	54
5.4.16.	Tool Plug-in API.....	54
5.4.17.	Tool Plug-in Toolkit API.....	55
5.5.	SECURITY.....	57
5.6.	PLUG-IN REGISTRATION.....	58
5.7.	CREATING A PLUG-IN “STEP-BY-STEP”.....	60
5.7.1.	Do I need a plug-in?.....	60
5.7.2.	What kind of plug-in shall I choose?.....	60
5.7.3.	Do I need a java plug-in to specify job input parameters?.....	60
5.7.4.	Implementing a plug-in.....	61
5.7.5.	Creating an OSGi bundle.....	61
5.7.6.	Signing a bundle.....	62
5.7.7.	Publishing a plug-in.....	62
5.8.	GUIDELINES FOR PLUG-IN DEVELOPERS.....	62
5.8.1.	Java plug-ins.....	62
5.8.2.	Using XML schema description.....	63
<b>6.</b>	<b>APPENDIX.....</b>	<b>64</b>
6.1.	INTERACTIVE APPLICATION USE CASE - TECHNICAL DETAILS.....	64
6.2.	PLUGIN REGISTRATION XML SCHEMA.....	66
6.3.	EXAMPLE OF XML SCHEMA JOB INPUT PANEL SPECIFICATION.....	68
6.4.	EXAMPLE OF JOB INPUT “READY-TO-USE” PLUGIN.....	72



## 1. INTRODUCTION

### 1.1. PURPOSE OF THE DOCUMENT

This document presents the results of analysis of BalticGrid application integration possibilities with the Migrating Desktop. As a result integration procedures and examples have been presented. Based on this document application developer can take a decision whether to integrate the following presented procedures.

### 1.2. ABBREVIATIONS

BG	– BalticGrid
G-PM	– Grid Performance Measurement tool
HEP	– High-Energy Physics
MD	– Migrating Desktop
MS	– Material Sciences
OCM-G	– OMIS-Compliant Monitor for the Grid
MPI	– Message Passing Interface
RAS	– Roaming Access Server
HPC	– High Performance Computing
OSGi	– Open Services Gateway initiative
XML	– Extensible Markup Language
SE	– Storage Element
LB	– Logging and Bookkeeping
RB	– Resource Broker
JAR	- Java Archive File
API	– Application Programming Interface
GUI	– Graphical User Interface
TLS	- Transport Layer Security
SSL	- Secure Socket Layer
CA	- Certificate Authority
VO	– Virtual Organisation
XSD	- XML Schema Definition
VNC	- Virtual Network Computing
EDG	- European DataGrid
WN	– Worker Node
BI	- Bioinformatics
IP	- Integration Point



## 2. EXECUTIVE SUMMARY

This document presents the BalticGrid applications analysis concerning their requirements of graphical interface to grid resources and overview of the Migrating Desktop (MD) framework functionality as the answer to the applications needs. The analysis of integration possibilities will be a base for applications developers to make a decision concerning integration. This document gives also important feedback to the Migrating Desktop developers concerning new requirements not yet supported.

The Migrating Desktop provides scientists with a framework which hides the details of Grid environment and allows for setting up and interactively controlling complex systems. This tool was designed and developed as the key product of the EU CrossGrid project. The BalticGrid will use the Migrating Desktop (MD) as a common point for accessing and managing the project applications, tools, resources and services.

The BalticGrid applications are dedicated to wide spectrum of various disciplines of science such as:

- bioinformatic - which requires sharing data from many sources and a diverse set of tools.
- material science - including atomic and molecular structures, modeling of advanced technological materials,
- high-energy physics - including statistical data analysis, production of Monte Carlo samples and distributed data analysis, nuclear and sub-nuclear physics and multi-body problems

The document aims to provide technical description of integration procedures between the Migrating Desktop and applications. The technical analysis leads to classification of applications regarding to features required by GUI side for grid operations. This includes pre-processing phase, job preparation, required input, job submission parameters and requirements, job types like batch or interactive, sequential or parallel, job status monitoring, processing the partial results, output definition and processing, security issues, multi-job processing, job dependency.

Possibilities and technical aspects of integration are presented and described for each feature. This is presented from user perspective and integrator/application developer perspective. Some set of applications (that requirements are accomplished by MD current functionality) can be integrated without any additional effort. In other cases application developers have to add appropriate plug-ins to fulfil application's specific needs. The concept of plug-ins as integration points between the Migrating Desktop and applications is being presented.

The performed analysis allows also to identify the requirements that are not supported by current version of the Migrating Desktop. For the requirements that are important from applications point of view additional work has to be done: based on detailed analysis and cooperation with application developers, proposition of missing functionality will be designed and presented in the next phases of the project.



### 3. BALTICGRID APPLICATION REQUIREMENTS ANALISYS

#### 3.1. BALTICGRID APPLICATIONS

##### 3.1.1. Analyzed applications

The analysis included in this document is based on the descriptions prepared by the developers of the following applications:

- Baltic Sea eco-system modelling
  - Finite element hydrodynamic-ecological model (SHYFEM)
  - Simulating WAVes Nearshore (SWAN)
- Bioinformatics
  - Collaborative Computational Project 4 (CCP4)
  - Crystallography and NMR system (CNS)
  - mpiBLAST
- High Energy Physics
  - Full simulation program for LHCb experiment. Monte Carlo data production (SimHEP)
  - Statistical Data Analysis. Estimation of the expected error of an indirect measurement (StatHEP)
  - Large scale Monte Carlo simulation of Ising model (MCIsing)
- Material Sciences
  - The General Atomic and Molecular Electronic Structure System (GAMESS) - 3 Application Descriptions
  - Gaussian03 (G03)
  - VPSM
  - ATOM
  - Large scale computer modelling of the kinetics of metastable systems with application to advanced electronic (ferroelectric) materials (SYMPLECTIC1)
- Modelling and simulation of heterogeneous processes in chemistry, biochemistry, geochemistry, electrochemistry, biology, and engineering
  - TestGen
  - SYNTSPEC
  - Optimal Design (Opt)
  - DEMPARALLEL (DEMPAR)
  - Scatter solver (Scatter)
  - ShgSolve
  - Numerical simulations of multicomponent biogeochemical contamination transport (NSBGT)
  - Numerical calculation of the distribution of loaded particles (LOAD)
  - SMEFLUX
  - Abinit
  - Domain Decomposition on Unstructured Grids (DOUG)



- Text annotation service
  - SentiKamols

### **3.1.2. Other applications**

Applications not taken into consideration:

#### **The Compact Muon Solenoid (CMS)**

Explanation: there will be no integration of CMS and the Migrating Desktop. This has been decided in the proposal phase.

#### **Aligns Nucleic Acid Conserved Elements (AlignACE), The MEME System (MEME), Pratt, SPEXS, Trie\*agrep family (Trieagrep)**

Explanation: most of the tools are just scripts/small C programs that might be combined in different ways which are much more comfortable to perform in a script. These bioinf tools are all meant for batch execution; besides, the potential users feel very comfortable with the command line, therefore it makes no sense to integrate such tools with MD.

#### **BiomedMining, SFSEARCH, Solution of the front movement (SFM)**

Explanation: The Application Description has not been provided for this application. There has not been also any other declaration from the application developers

## **3.2. APPLICATION REQUIREMENTS**



**PROCEDURES OF INTEGRATING APPLICATIONS WITH  
THE MIGRATING DESKTOP**

	<b>Mode</b>	<b>preprocessing</b>		<b>input</b>		<b>metadata</b>	<b>output</b>		<b>visualization</b>	
			<b>format</b>	<b>parameters</b>	<b>files/ direct ories</b>		<b>format</b>	<b>Partially result</b>	<b>type</b>	<b>Java?</b>
SHYFEM	Batch	Text editor is used. Some files are converted to binary format by utilities	10,25 files, even 500 MB	No	yes	no	About 80 binary ASCII files (not more than 1 GB)	no	no	No
SWAN	batch	Text editor	5 Files	No	yes	no	Files,ASCII, binary	No	no	No
CCP4	batch	Manual editing of the scripts, perl-based generators, tcl/tk-based script generators	Files, pdb, ascii, binary MTZ,	yes	yes	?	MTZ, PDB, PDB format data, ASCII, line-oriented files 5-50 MB	Can be useful, format the same as output	pdb	No
CNS	batch	Manual editing of the script, perl-based generators, Web form-based generators	PDB format, ASCII, line-oriented, FOB format, or binary MTZ reflection file, converted to FOB on the fly with CCP4 program suite, other	yes	yes	?	PDB format data, ASCII, line-oriented 5-50MB	Can be useful, format the same as output	PDB format	No
mpiBLAST	batch	none	Formatted database, file	yes	yes	?	Search result, txt	Yes, format the same as output files	No, web interface	No
SimHEP - GAUSS	Batch	none	Many files with configuration parameters and options	Yes	Yes/yes	No	Raw data 50-1000 MB	no	no	No
SimHEP - BRUNEL	Batch	No	Raw data produced by GAUSS module	Yes	yes	No	File, ESD data 100-500MB	no	no	No
StatHEP	Batch	Simple script to generate a series of files containing input params	Ascii file, for one production run series of input files has to be generated	Parameter files	yes	No	Ascii files with histograms	maybe useful	Graphics provided in ROOT	Can be written
MCIcing	Batch	Read-write instructions for data file	File, params	From file	No	No	Several files, text files	Yes, binary	Yes,xmgrace	Can be rewritten



**PROCEDURES OF INTEGRATING APPLICATIONS WITH  
THE MIGRATING DESKTOP**

Gaussian03	Batch	From application	File, gjf	yes	yes	No	yes	no	yes	Yes
GAMESS a	Batch	From application	File, gjf	yes	yes	No	Files, text	no	YES	Yes
GAMESS b	Batch	Manual preparation of input file	file	yes	yes	No	File, up to 2 GB	Yes subsequent text output	Yes, Molden or Molekel programs	No need
GAMESS c	Batch	Text file	File, text	?	?	No	File or stream, text, binary	?	Desirable but not necessary	?
VPSM	Batch	manual preparation of the input file	yes	yes	Yes temp dir in code	no	Text, binary ,up to 1Mb size. The solutions (kept locally) which may be reused in subsequent runs for additional information take several Gb. Intermediate files (stored locally) take several Gb, depending on a problem,	Yes, subsequent text output	no	No
ATOM - AS	Batch	Preparation of input files	file	yes	yes	yes	Files, text and binary	Yes, text, binary	no	No
ATOM - GLOCHAS	Batch	Manual preparation of input files	files	Yes	yes	No	Files, ASCII, average to 30 GB	Yes, ASCII	no	No
ATOM - POLARIZ	Batch	no	Several files	yes	yes	No	Formatted data file	no	no	No
SYMPLECTIC1	?	A small amount of material constants, driving data, and electric and elastic boundary conditions are fed at hand.	File, manual input	yes	no	no	File, text, images	yes	Images, graphics	No
TestGen	Batch, Some versions may be stopped manually	Settings file (created manually or with the help of custom script/program),	Text files, previous results, previous state of files	yes	yes	no	Test patterns: , fault model status (transfer matrix, term lists, etc.) and other	Yes (in some cases): text to stdout or stderr describing current	No, for some WWW interface	No



**PROCEDURES OF INTEGRATING APPLICATIONS WITH  
THE MIGRATING DESKTOP**

	without losing results,	optional results from previous runs (may be merged, reversed order and so on, done by user manually).					(depend on application version)	step or progress		
SYNTSPEC	Batch	Aut. Gen. input	File, ASCII	From file	Yes	No	File, text	No	No, vis. Outside GRID	No
Optimal Design (Opt)	Batch	no	File	yes	yes	no	Text and images	Sometimes yes	Yes images if needed	Can be written
DEMPAR	Batch	Included in domain code	2 files	yes	no	no	5 MB file,ascii	no	Python GUI	Yes
Scatter	Batch	no	input file contains initial light beam properties, program arguments specify scattering media properties	parameters are passed through command line	yes	no	File, 2-3 MB, maximal: 80 MB	Yes, partial results are printed to console	no	no
ShgSolve	Batch	it may be necessary to calculate a fixed set of scalar double-typed expressions	File with params	yes	no	no	300-2GB files in fixed directory structure	Yes. Several text files. 1D arrays in text files. Rectangular 2D arrays in text files.	gnuplot	Yes, possible to rewrite
NSBGT	Batch	Read-write instructions for data file	File, 1 KB	yes	no	no	Output is the result files (about 5 MB) and a temporary file to continue simulation from a record, which is about 100 MB depending on space and time grids. Text, binary	yes	Yes, xmgrace	Possible to rewrite to java
LOAD	Batch / The application during runtime presents partial results for various time moments and the user can	Input file is prepared by end-user	Application needs the parameters characterizing the process to be simulated. The file of parameters is about 10 kB. The parameters can be	yes	no	no	Numerical file 30-100kB	yes	Results are numerical but they can be presented graphically	It is possible to rewrite in Java



**PROCEDURES OF INTEGRATING APPLICATIONS WITH  
THE MIGRATING DESKTOP**

	control these moments and the number of the presented results interactively.		mutually dependent.							
SMEFLUX	Batch	Manual preparation of input file	file	yes	yes	no	Output files, text	no	Graphical visualization will be made without the GRID facilities	no
Abinit	Batch	no	file	Yes. file containing information about the solid to be modelled Application needs also atomic pseudopotential data files which may be stored, but not necessarily, in a separated directory.	yes	no	The final output is written into several text files.	During computation application writes the log text file and stores intermediate results into binary files needed to restart job in case of crash.	After computation I use gnuplot and xmgrace to present data	no
DOUG	Interactive, changing certain solution parameters of the running application.	- one or more input files (ASCII or binary) representing linear system as an output from, e.g., FEM, FDM or some other codes; - input ASCII file containing DOUG control parameters editable by a user in any text editor.	ASCII control file, ASCII or binary file(s) with linear equation system representation (integers and single or double precision numbers)	ASCII file (doug-control.file) contains a number of DOUG control parameters. Among them is/are path(s) to (ASCII or binary) file(s) with a linear equation system representation.	Yes/no	no	Application produces an output file containing a solution of a system of equations. Typically ASCII or binary representation. Size of solution file can vary from kilobytes to several tens of megabytes. Application can also produce ASCII file(s) containing logs of the solution process.	Partial results as a log to standard output. In debugging mode it is possible to use PLplot to visualise some intermediate results.	Partially, yes. Using PLplot it is possible to visualise some intermediate results	Possible to rewrite to java



**PROCEDURES OF INTEGRATING APPLICATIONS WITH  
THE MIGRATING DESKTOP**

---

SentiKamols	Yes,	Input is user-supplied through web interface	Database	Application receives user supplied text as input	yes	No, What do you mean?	Plain text mixed with HTML	no	no	no
-------------	------	--	----------	--	-----	-----------------------	----------------------------	----	----	----



**PROCEDURES OF INTEGRATING APPLICATIONS WITH  
THE MIGRATING DESKTOP**

	<b>Post processing</b>	<b>parallelization or distribution</b>	<b>Many jobs at once</b>	<b>Cascade of jobs</b>	<b>Security requirements</b>	<b>Example jdl</b>	<b>Grid interest</b>
SHYFEM	Matlab scripts and special utility independently of main routine	Parallelization	Yes for some tasks, independent	Yes for some tasks	no	no	Possibility to use computing resources to decrease computing time
SWAN	Possibility to import data to MatLab from *.mat file	MPI, OpenMPI	no	no	no	no	Possibility to use computing resources to decrease computing time
CCP4	Processed data enters the phase-build-refine cycle of the crystallographic data processing	Multiply runs	Yes, rather independent. Multiply runs with different input parameter sets	Mostly no	Unix security	Shortly yes	Grid will decrease response time when running multiply jobs, by running jobs in parallel on multiply machines. Large datasets can be easily parallelised
CNS	Processed data enters the phase-build-refine cycle of the crystallographic data processing	Multiply runs	Yes, rather independent. Multiply runs with different input parameter sets	Mostly no	Unix security	Shortly yes	We expect that using the GRID will decrease response time when running multiple jobs, by running jobs in parallel on multiple CPUs.
mpiBLAST	none	MPI	yes	no	no	no	?
SimHEP- GAUSS	Output processed by next module	Parallelization	Yes, independent jobs. Generation of la arge number of events is done by splitting into jobs each for 1000 events	No. Only cascades of subprocesses inside one job	Grid security	Yes available	Already used on grid
SimHEP- BRUNEL	no	Parallelization	Yes. Generation of a large number of events is done by splitting into jobs each for 1000 events	no	Grid security	Yes available	Already used on grid
StatHEP	Each job will produce the results for a given set of parameters	yes	Yes, independent jobs	no	No	Not yet	Get results in reasonable time
MCIIsing	visualisation	Hybrid OpenMPI and MPI	yes	yes	no	no	Possibilities of HPC
Gaussian03	GaussView program	SMP	yes	yes	no	no	Accelerate calculation
GAMESS a	many	DDI	yes	yes	no	no	Accelerate calculation
GAMESS b	Reading the outputs, some of them can be reused, also for restarting calculations	?	no	No/Yes	no	no	Big memory, larger CPU resources, faster investigations



**PROCEDURES OF INTEGRATING APPLICATIONS WITH  
THE MIGRATING DESKTOP**

GAMESS c	Analysis of output also via visualization tools	yes	Possible, several independent	no	no	no	?
VPSM	Reading of the output. Binary files can be reused to calculate additional observables	MPI, Each node calculates assigned tasks (pieces of acting a matrix on a vector, vector-vector operations) which are distributed by difficulty.	no	yes	no	no	big memory and CPU resources
ATOM - AS	Reading of the output, binary and text files can be reused to calculate additional observables	no	no	yes	no	no	larges RAM
ATOM - GLOCHAS	Reading of output files	Not yet	yes	yes	no	no	Big memory and CPU resources
ATOM - POLARIZ	No	no	no	yes	no	no	to make faster and shorter calculations
SYMPLECTIC1	Visualizing by MATHEMATICA tools in physical units.	no	no	no	no	no	Faster results, better performance, more detailed results
TestGen	Results are checked on commercial fault simulators. Other uses of results depend on particular implementation and required results.	Some versions may use custom protocols	Yes. Random data are analyzed, so to get usable results, repeated identical experiments are required. These runs are independent of each other. If user is using iterative pattern generation – there is a set of tasks to be over before next step can be started.	yes	no	Yes, example	Many similar independent runs required to get trusted results. Jobs schedule distributed not in linear fashion: at some intervals a big number of experiments are done, while at other time there are no jobs run.
SYNTSPEC	Collecting output files for visulisation	In progress	yes	yes	general	soon	Possibility to run several jobs, make cascades, use big storage
Optimal Design (Opt)		MPI, Message passing is used to distribute data and collect the results. Scaling up to 8 processors is tested.	no	no	no	no	GRID environment would be useful for resource concentration
DEMPAR	no	The parallelization of the code is based on the domain decomposition, which has been established as the most	sometimes	no	no	no	Actual DEAMPARALLEL applications need a lot of resources. In Baltic countries growing GRID infrastructure is able to provide them.



**PROCEDURES OF INTEGRATING APPLICATIONS WITH  
THE MIGRATING DESKTOP**

		efficient strategy for distributed memory PC clusters. Inter-processor communication is performed by the message passing library MPI					
Scatter	no	no	Yes, independent	sometimes	no	no	-
ShgSolve	generation of 2D graphs using gnuplot	no	No, not depended	Maybe	no	No	-
NSBGT	Visualizing by xmgrace	MPI	no	No	no	no	Possibilities of HPC (parallel computing)
LOAD	The output is received in a numerical form and the user can choose his/her own processing form.	no	yes	no	no	no	The jobs can be computed independently and their number is crucial for understanding the processes to be simulated
SMEFLUX	It is necessary to collect the output files for visualizations and analysis	Work is in progress	yes	yes	general	Not yet	CPU resources, storage, RAM, possibility to run several jobs
Abinit	no	Work in cluster	no	no	no	no	?
DOUG	The application by itself (or the chosen solution method) can be analyzed e.g. by plotting solution convergence graph. (PLplot)  Solution can be visualized or tested by a user separately.	MPI, tried on 32 proc. in cluster	no	no	no	no	GRID integrates clusters of computers and gives access to them. Users using GRID-enabled DOUG can benefit from this infrastructure.
SentiKamols	Ouptut is shown to user in a web page	Application uses XGrid for distribution and parallelization	Yes, independent	no	Prolog source code not accessible by unrelated parties	no	The jobs are easily parallelizable and there are lots of texts to process



### 3.3. APPLICATION SPECIFIC REMARKS

#### **GameSS a**

*Large and small problems should be calculated separately to avoid long waiting*

#### **STATHep**

*User interface is needed to generate input files and submit a bunch of jobs into a Grid.*

#### **ATOM GLOCHAS**

*Different branches of cascades could be distributed on different machines*

#### **SemtiKamols**

*It would be good to find a possibility to get machine useable data (as opposed to human readable) from grid middleware software (like job ID, status, errors etc.). Ideally it would be nice if the grid middleware functionality were accessible through a dynamically loadable library written in C (for easy integration with software that is not maintained by grid middleware software developers).*

#### **VPSM**

*The operation of matrix acting on a vector takes the most computation time. Effective work balance, taking into account data transfer delays could be an issue. Due to scarcity and big dimensions there are a lot of "store" operations reducing the effectiveness of the CPU usage.*

#### **TestGEM**

*make use of OS and batch system features to exit without losing the results in case of termination at expired limits.*

*use of core dumps for problem identification.*

*debugging runs (without randomization) to identify possible problems on different clusters.*

#### **DEMPAR**

*Employed GRID should support local MPI installations or MPI-G2.*

#### **DOUG**



*Necessity of a simple (e.g., Java) interface that a user can launch locally or remotely from the DOUG-server (e.g., via JNLP).*

*The general part of the interface must contain the possibility to set DOUG control parameters and specify paths to files representing a system of equations and where to store a solution (in general, file names do not necessarily need to be local to the submitting machine). The interface could allow for running DOUG on:*

*1. a local machine or a cluster*

*2. GRID*

*2.1 DOUG is pre-installed on a number of clusters on the GRID*

*2.2 The GRID-cluster allows for compilation in place. Then at job stage-in before running a task DOUG is fetched from a repository (or grid storage element) and compiled. (Job description must be altered by interface accordingly.)*

*If, after specification of control parameters and paths to files, option two was chosen ("run on GRID"):*

*1 interface*

*1.1 asks a user to present their grid proxy (stored locally or in a proxy server)*

*1.2 asks the user whether he/she wants to receive e-mail notification about job statuses (scheduled, started, finished, aborted, etc.) - relevant for long running tasks*

*1.3 based on job specification writes down a job description in an appropriate language*

*1.4 submits the task for execution:*

*1.4.1 directly sends job to GRID if*

*-- there is an (Java) API for doing that from within the interface*

*or*

*-- the submitting node has the necessary software (some UI) installed on it*

*1.4.2 otherwise, interface can send a job description (or only DOUG control parameters) and user's proxy to the DOUG-server which can have all necessary software for submitting a job for calculation installed. In this case job submission can potentially be done to any GRID based on different middlewares (e.g., LCG, NorduGrid ARC), supposing the user is authorized to submit jobs to them.*

*NB! in both previous cases (1.4.1, 1.4.2) if files describing an equation system are stored on a user machine, the interface must offer a possibility to move them into a GRID storage element (where the user is authorized) and the interface must change the job description accordingly, i.e., at job stage-in at a GRID-enabled cluster input files must be fetched from the storage element.*

*1.5 after successful job submission the interface returns ID of a grid job and stores it locally or on the DOUG-server for the subsequent interaction with the GRID information system for querying the status of the job.*



*NB! (extra feature) The latter case with storing e.g. grid job's ID on the DOUG-server (in some database) introduces certain service for the registered users. Such a service on the DOUG-server side can include the storing of complete histories of tasks (job descriptions, links to input/solution files stored on GRID storage elements, etc.) so the user can examine them and, if necessary, compare results and/or re-run certain tasks.*

*2 at this point, if interactivity is not taken into account, the user can close the interface and either wait for the notification stating the end of calculations or some time later launch the interface again and check the status of the job (or if the user has registered with the DOUG-server, view the job status on a web via tasks tracking interface on the server).*

*3 if the job has finished, download result(s) on a local computer.*

*Interactivity between steps 2 and 3.*

*a. DOUG->user:*

*\* visual representation of some intermediate solution results:*

*-- convergence graph (a double precision number after each iteration)*

*-- ...*

*\* technical information about the running program (possibly presented visually):*

*-- load of computing nodes*

*-- ...*

*b. user->DOUG:*

*\* change certain solution parameters of the running application*

*Some notes on implementation of interactivity.*

*\* For long running tasks the user must have a possibility to close the interface, then launch? it again (possibly, from another computer) and connect to the application (can be implemented via the DOUG-server (?)). This must not affect the functionality offered by the interface.*

### **3.4. APPLICATION REQUIREMENTS REVIEW**

#### **3.4.1. Application input**

Most applications use one or more files (up to 100) or whole directories as an input. Files are in different formats, the sizes of files are up to 500MB. Also most applications have specific parameters, some of them use files that contain sets of parameters.

The input for SentiKamols, mpiBLAST comes from databases.



### 3.4.2. Preparing the input

Some of the applications use scripts, web or perl generators for creating files containing input data; however, in most cases the end-user prepares input files manually, using text editors.

Methods of preparing the input	Applications
Text editors	SHYFEM, SWAN, SSP4, CNS, , GAMESS b, GAMESS c, VPSM, ATOM AS, ATOM GLOCHAS, SYMPLECTIC1, LOAD, SMEFLUX, DOUG
Script generators	SSP4, StatHEP, TestGen
Web-based generators	CNS, SemtiKamols
External applications	Gaussian03, GAMESS a, DEMPAN, DOUG, ShgSolve (calculations)
Other	SYNTSPEC

### 3.4.3. Job submission

### 3.4.4. Parallelisation

The significant number of BalticGrid applications runs on clusters using some parallelization mechanisms (the most common is MPI)

### 3.4.5. Running modes

Most BalticGrid applications are running in a batch mode but some of them require interaction with the user while computing results. In the interactive mode the user should have a possibility to change parameters of the job during runtime, and get the response in near real-time. The interactive job should run immediately and the user should have access to bidirectional channel to/from the job. The user should have also possibility of stopping applications without losing results and to track partial results as they are produced by the application.

### 3.4.6. Multiply jobs and their interdependencies

Many applications require the possibility of submitting many jobs at once. This requirement results of the following reasons:

- for some purposes the user has to submit many jobs which differ from each other in only a few input parameters (that in most cases comes from a well-defined set of values)
- sometimes the results of one job are reused as an input for the next one (jobs create a so called “cascade of jobs”). In this case the order of sending jobs is important, especially in case of failure.

Casades: MCIing, Gaussian03, GAMESS a, GAMESS b, VPSM, ATOM-AS, ATOM-GLOCHAS, ATOM-POLARIZ, TestGen, SYNTSPEC, SMEFLUX, SHYFEM, ShgSolve, Scatter



The following application requires submitting many independent jobs at once:

LOAD, SMEFLUX, SYNTSPEC, TestGen (repeated experiments), ATOM – GLOCHAS, GAMESS – c, GAMESS – a, Gaussian03, MCIsing, StatHEP, SimHEP – BRUNEL, SimHEP- GAUSS, mpiBLAST, CNS, CCP4, ShgSolve, DEMP PAR, SCATTER

### 3.4.7. Output format

The output of application can be stored in several files (up to 100), the size of outputs could be up to 30 GB. Formats of output files are usually: ASCII, MTZ, PDB, ESD, binary. In one case output could be a stream (GAMESS c), and in another case output is in the HTML format (SentiKamols)

### 3.4.8. Partial results

There are different goals and motivation for showing partial results. Some applications present standard output or error for logging information or to check current step or progress. Others visualise such results for debugging or for taking the decision whether to stop the job or not.

The following applications use partial results:

CCP4, CNS, mpiblast, StatHEP, MCIsing, GAMESS b, VPSM, ATOM AS, ATOM – GLOCHAS, SYMPLECTIC1, TestGen, Opt, NSBGT, LOAD, Abinit, DOUG, ShgSolve, SCATTER

### 3.4.9. Visualisation

Most of the analyzed applications produce text or binary files as their results, but some of them use some visualization methods to present the results in a more “user-friendly” manner. The BalticGrid applications use various kinds of visualization:

Visualisation mechanism	Applications
WebInterface:	mpiblast, TestGen
PDB	CCP4, CNS
ROOT:	StatHEP
Xmgrace	MCIsing, NSBGT, Abinit
Gnuplot	Abinit, shgSolve
Plplot	DOUG
Images	SYMPLECTIC1, Opt
Phyton GUI:	DEMPAR
Java visualisation	GAMESS03, GAMESS a, DEMP PAR.
Other	Gaussian03, GAMESS a, GAMESS b, SYNTSPEC, SMEFLUX



The following application could have visualization in Java if there is such a need: StatHEP, Opt, NSBGT, LOAD, DOUG, ShgSolve

#### **3.4.10. Processing results**

The post-processing phase is usually done with the help of visualization tools like xmgrace, gnuplot, plPlot, GaussView or by application like Matlab, commercial fault simulator (used to evaluate the attained fault coverage by TestGen application) or others.

For some applications the post-processing phase is done by the next module in chain, and the results are reused.

### **3.5. SECURITY REQUIREMENTS**

There are no special security requirements. The current grid level of security seems to be enough. Some applications (CCP4, CNS) pointed to Unix security.



## 4. INTEGRATION WITH THE MIGRATING DESKTOP

### 4.1. MIGRATING DESKTOP OVERVIEW

The Migrating Desktop (<http://desktop.psn.c.pl>) was designed and developed by Poznan Supercomputing and Networking Center in close collaboration with other partners within the EU CrossGrid project (IST-2001-32243) (<http://www.crossgrid.org>). As the key product of that project, the Migrating Desktop has proved its usefulness in everyday work of the CrossGrid community. It is an advanced graphical user interface and a set of tools combined

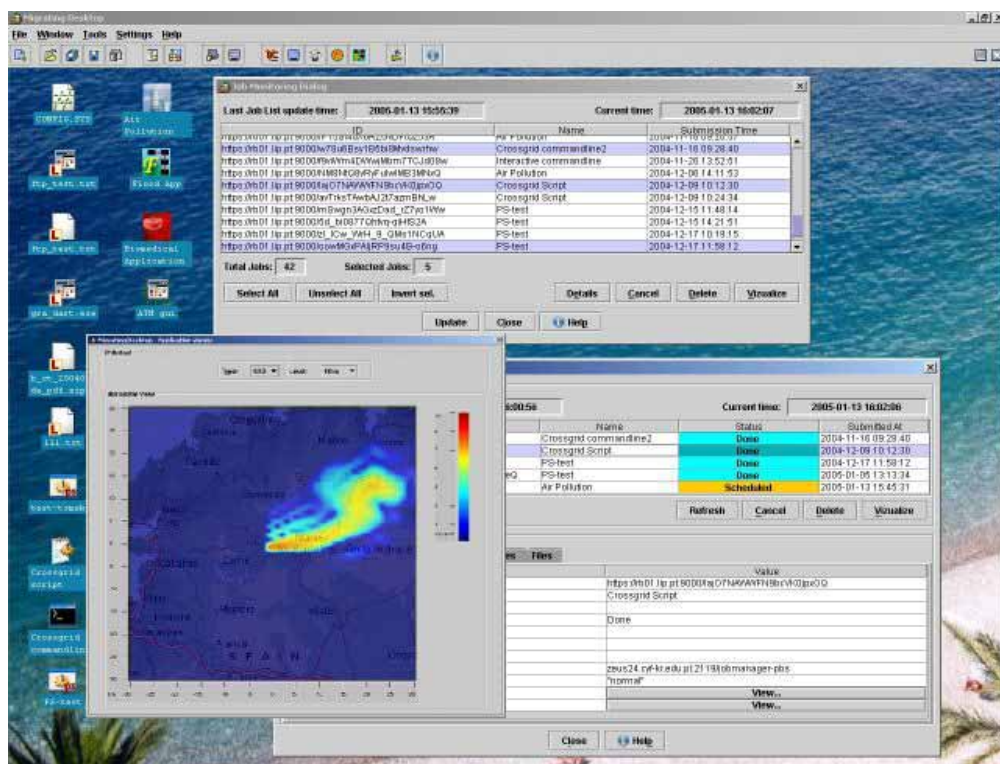


Fig. 1 Migrating Desktop main window

with a user-friendly outlook, similar to a window-based operating system that hides the complexity of the grid middleware and makes access to the grid resources easy and transparent. The Migrating Desktop offers: a flexible personalized working environment available independently of the user location, scalability and portability, a set of tools, a single sign-on mechanism, support for multiple grid infrastructure and special support for „roaming users”, so that they can use their personalized working environment regardless of their physical location and used operating system. The Migrating Desktop strictly cooperates with the Remote Access Server (RAS), which intermediates between different grid middleware and applications. The RAS offers a well-defined set of web-services that can be used as an interface for accessing HPC systems and services (based on various technologies) in a common, standardized way.



The key feature of the Migrating Desktop is the possibility of easy adding various tools, applications and support visualisation of different formats. Due to the complex nature of grid applications and unpredictability of their requirements, the Migrating Desktop offers a framework that can be easily extended on the basis of a set of well-defined plug-ins used for: accessing tools, defining job parameters, pre-processing job parameters before submission, and visualisation of job results. It makes that product significantly more flexible than specialized tools (e.g. portals) designed only for a specific application.

The Migrating Desktop framework architecture is designed on the basis of a concept of OSGi specification designed by The OSGi Alliance (<http://www.osgi.org/>). The Migrating Desktop follows OSGi Service Platform specification and is based on the same plug-in engine as the Eclipse platform (Equinox, <http://www.eclipse.org/equinox/>) that provides a general-purpose, secure, and managed Java framework that supports discovering, integrating, and running modules called bundles. Open architecture of the Migrating Desktop makes integration with various tools, applications and middleware extremely easy (in contrast to other products that are only Globus-oriented in most cases). Such approach allows increasing functionality in an easy way without the need of architecture changes.

## **4.2. DESCRIPTION OF THE MIGRATING DESKTOP FUNCTIONALITY WITH RESPECT TO THE BALTIC GRID APPLICATION REQUIREMENTS**

### **4.2.1. Application input**

The user can specify inputs using the Job Submission Wizard. This Wizard is responsible for proper preparation of the user's job input parameters and consists of several panels. One panel is an application-specific plug-in that can be used for defining parameters specific for a given application, getting parameters from a database and other application-specific sources. The rest of panels can be used to set common parameters such as: job information, resource requirements, files and environment variables.

- Specific application parameters panel - application plugin (Integration Point 1 (IP1)) or XML (Integration Point 2 (IP2))
- Job description
- Resources requirements (common like job type, memory, CPU, etc.)
- Input and output files – the user can choose one of more file. The outputs are created physically before submitting the job (empty files).
- Environment variables
- Pre-processing tools like OCM-G (Integration Point 3 (IP3))

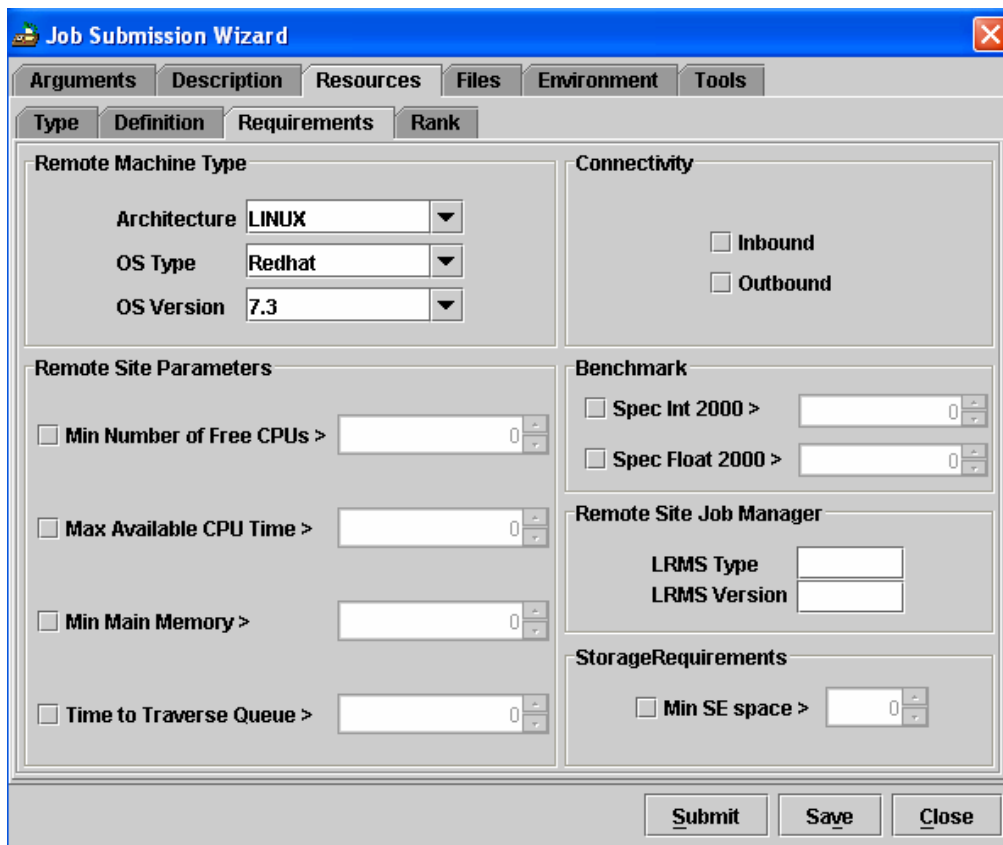


Fig. 2 Job Submission Wizard

At the moment there is no automatic way for selecting directory as an input or output. As it is one of the BalticGrid application requirements, such mechanism shall be prepared.

There are no constraints concerning the input files size or the number of these files.

#### 4.2.2. PREPARING INPUT

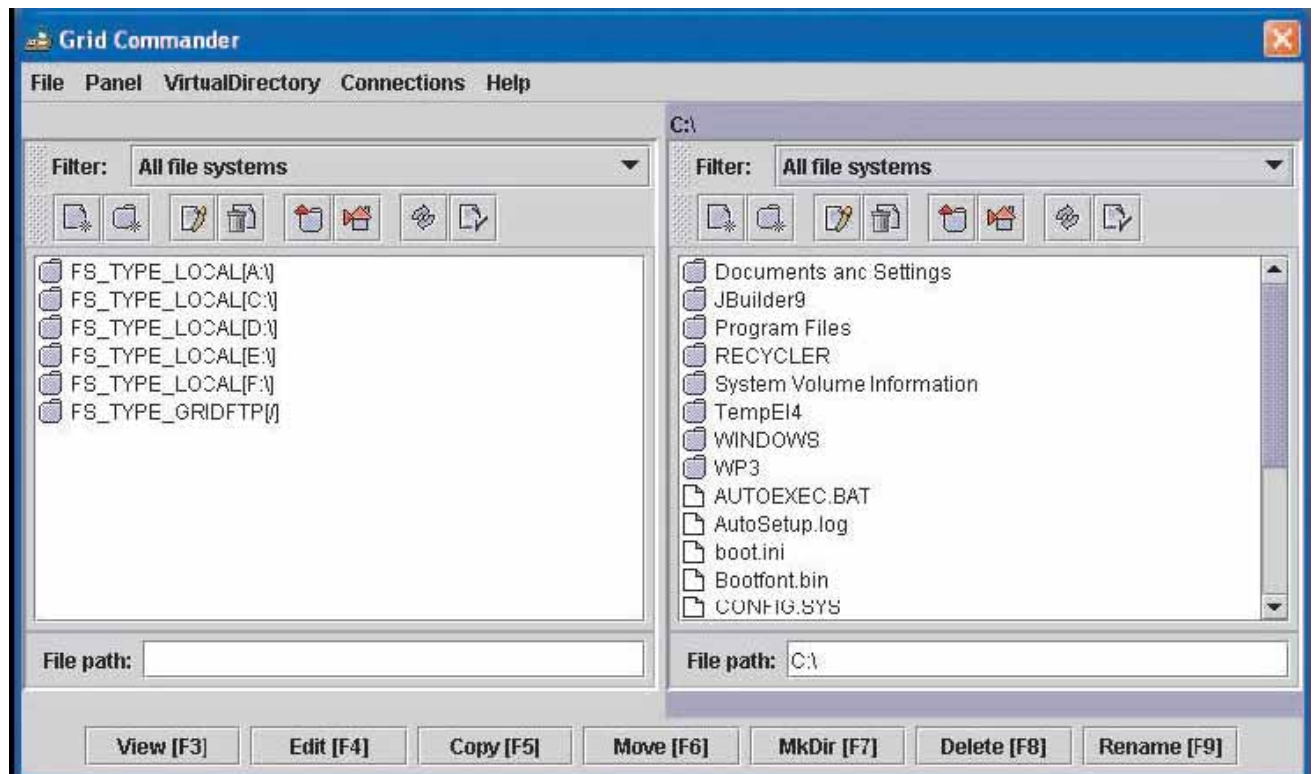
The pre-processing phase can be done locally using applications or tools installed or available locally. A set of files is usually prepared during the preprocessing phase. Such files can be copied from a local workstation to the grid using a file manager provided by the Migrating Desktop (so called GridCommander). In some cases job parameters defined by user needs to be preprocessed (eg. to add some additional parameters for debugging the application etc). For that purposes user may use set of job processing tools defined as the Migrating Desktop framework plug-ins and available in Job Submission Wizard. Example of such tool can be grid application monitoring system (OCM-G) developed within the BalticGrid project.

##### File management using GridCommander

The Grid Commander is a two-panel application similar to the Commander family tools. A single panel can represent a local directory, gridftp or ftp directory, or other protocol to the native storage. Each protocol is defined as a plug-in that makes file management in MD easy to extend to other protocols. From the technical point of view graphical elements operate on



generic file systems that invoke appropriate functions within the supported plug-in the User Virtual Directory. It is an abstract file-system that contains information about all user files independently of their physical location. Each branch in the Virtual Directory tree can be physically placed on a different location or can even be placed on storage with a different way of accessing (e.g. ftp, gridftp or any other project native protocol). The Virtual Directory was designed to standardise data access, and to create a user-friendly, uniform view of the Grid and local files.



**Fig. 3 Grid Commander**

The Migrating Desktop also contains a basic text editor so some simple scripts could be manually changed directly on SE or gridFTP servers.

#### **Submission of jobs with OCM-G support.**

OCM-G is a grid application monitoring system. The system was developed in the CrossGrid project and is being adapted to gLite middleware in the frame of the BalticGrid project. The main motivation of this is to help application developers to adapt their application to the GRID environment by enabling possibilities to study performance on-line.

Enabling monitoring with this system require some special additions to the job in the process of submitting it. For this purposes a special plug-in was implemented to enable spawning OCM-G main monitor process and to make required additions to the job description. In the frame of this project the plugin will be maintained and reimplemented to the new version of job submission API if necessary .



### 4.2.3. Job submission

The Migrating Desktop supports several types of jobs: batch or interactive, sequential or MPI (MPICH-G2 and MPICH P4).

Some features of the Migrating Desktop like MPI or interactivity require appropriate support on the grid middleware level (RB, CE).

### 4.2.4. Running modes

#### a. Batch jobs (sequential or MPI)

Integration points and possibilities shall be presented below on submitting example batch application use case

1. To submit example batch jobs the user starts the Application Wizard, then chooses test, PS job.
2. The Job Submission Dialog will open. This wizard simplifies the process of specifying parameters and limits, suggesting user defaults or recently used parameters. The Wizard is responsible for proper preparation of the user's job and consists of several panels.
3. The user can submit the job immediately or choose options on several tabs:
  - a. Specific application parameters panel - application plugin (Integration Point 1 (IP1)) or XML (Integration Point 2 (IP2))
  - b. Job description
  - c. Resources requirements
    - i. jobType = "normal"(default) for batch jobs
    - ii. jobType = "mpich" for batch, MPICH-P4
    - iii. jobType = "mpich-g2" for batch, mpich-g2
  - d. Input and output files
  - e. Environment variables
  - f. Preprocessing tools like OCM-G (Integrating Point 3 (IP3))
4. After submission the user will get confirmation about the submitted job.
5. The user will check the status of the job/details/outputs in the monitoring dialog. He/she can check details that come from LB. The user can check partial results (description in section partial results). He/she can visualize output files separately if there is an appropriate visualization plug-in (could be default already included in MD) (Integrating Point 4 (IP4)) or visualize the application using several files with a special visualisation plug-in (Integrating Point 5 (IP5)).

#### b. Interactive jobs

The requirements concerning the applications that declared themselves as running "interactively" as presenting partial results during runtime (LOAD) and stopping application without losing results (TestGen) are available also in applications running in batch mode.

Only DOUG application requires real interaction with the user to change application parameters during execution.



Interactivity mechanisms developed in the CrossGrid project had enhanced the interactivity approach developed in the DataGrid project. There are two different kinds of client applications: the Java visualizations client that can be plugged into the Migrating Desktop, and a legacy application that cannot be ported in Java to be integrated as a new plug-in into MD, and that will run in an .under user control machine which we named the Application Client Machine. (ACM). In order to show the output of a job running on a remote machine we use the VNC protocol.

Concerning the BalticGrid, if we want to support such kind of applications, we will have to set up additional RB + install additional software on CE.

Integration points and possibilities will be presented below as an example of interactive application use case

1. To submit example interactive jobs the user starts the Application Wizard, then chooses “test”, “Interactive commandline”.
2. The Job Submission Dialog will open. This wizard simplifies the process of specifying parameters and limits, suggesting user defaults or recently used parameters. The Wizard is responsible for proper preparation of the user's job and consists of several panels.
3. The user can submit the job immediately or choose options on several tabs:
  - a. Specific application parameters panel - application plugin (Integrating Point 1 (IP1)) or XML (Integrating Point 2 (IP2))
  - b. Job description
  - c. Resources requirements
    - i. jobType = “interactive” for interactive jobs
    - ii. jobType = “interactive, mpich” for interactive, MPICH-P4
    - iii. jobType = “interactive, mpich-g2” for interactive, mpich-g2
  - d. Input and output files (StdInput, StdOutput, StdError must be null)
  - e. Environment variables
  - f. Preprocessing tools like OCM-G (Integrating Point 3 (IP3))
4. After submission the user will get confirmation about the submitted job.
5. The job should start almost immediately, thanks to the Condor Bypass mechanism
6. The user will check the status of the job/details/outputs in the monitoring dialog. He/she can check details coming from LB. The user can run the interactive plugin (Integrating Point 6 (IP6)). Such plugin can use API to retrieve a bidirectional channel of input and output streams to and from the application. In such a way it is possible to control the application while it is running sending input parameters.

Whole interactive use-case with technical information may be found in Appendix 1.

#### **4.2.5. Multiply jobs and their interdependencies**

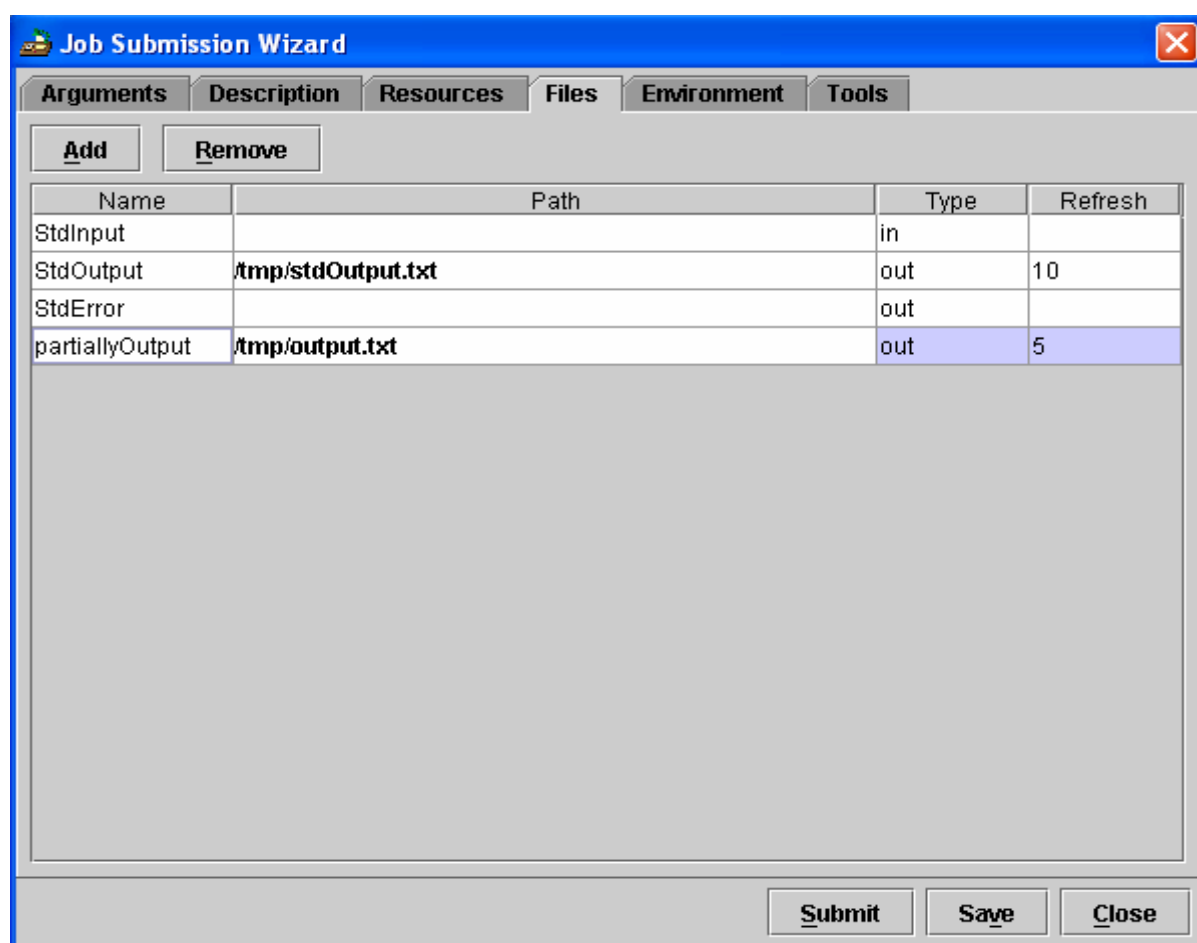
A part of applications requires a definition of a set of jobs and (sometimes) interdependencies between them (to proceed input of one job as output for another one). At that moment the Migrating Desktop



system does not support this feature. Due to the importance of that functionality for many BalticGrid applications some activities will start. Those activities shall result in the integration of the existing graphical tool for defining multiply jobs and dependencies between them within MD in the nearest future.

#### 4.2.6. Partial results

A special mechanism has been prepared to present partial results of the running application. During the submission of a job, the job submission plug-in or the user should specify output files with the option “Refresh” and give a refresh period. All output files that are refreshable should be created before the submission of a job (creating on SE empty files).



**Fig. 4 Job Submission Wizard – choosing files**

When job runs on some CE, a special script is copying every refresh period time of the output files that are created locally on the CE to the localization specified during job submission. Such file/files can be visualized by the visualization plug-ins available in the Migrating Desktop like text viewer or jpg/svg viewer or Application developers can write their own visualization plug-ins (Integrating Point 4 (IP4)).



#### **4.2.7. Processing the output**

#### **4.2.8. Visualisation**

The Migrating Desktop offers strong support for the visualization of application results. The user can visualize several standard format files (eg. Jpg, bmp, svg, txt, etc.) using built-in file viewers. It is also possible to see the partial result using automatically refreshable file viewers or use external visualization tools installed on a local system. Every application that needs more sophisticated visualization than viewing its output files can provide its own visualization plug-in. The built-in set of viewers can also be easily extended by application developers using the idea of plug-ins.

#### **4.2.9. Security requirements**

The Migrating Desktop follows standard grid security mechanisms based on certificates and delegations with all its advantages and weaknesses.

### **4.3. COMMENTS TO “SPECIFIC APPLICATION REMARKS”**

This section presents only additional remarks for applications.

DOUG application described whole possible use-case of running their application on the grid. It is presented below how Migrating Desktop can simplify and support this use-case.

The Migrating Desktop fulfils most of DOUG application developer expectations:

- asks a user to present its grid proxy (stored locally or in a proxy server) (DOUG1.1)
- based on job specification writes down job description in appropriate language (DOUG1.3)
- directly sends job to GRID (DOUG 1.4.1)
- it is possible to move files/results to/from some GRID storage element (DOUG1.4)
- after successful job submission interface returns ID of a grid job and stores it in profile (DOUG1.5)
- all history of user jobs is stored in database with all details like job descriptions, links to input/solution files stored on GRID storage elements, etc. (DOUG1.5)
- user can download results locally (DOUG3)
- user can check results while job is running (DOUG2a)
- it is possible to visualise intermediate solution results (DOUG2a)
- it is possible also to change the parameters while the application is running (DOUG2b)
- long running tasks user have a possibility to close the interface, then lunch it again (possibly, from another computer) and connect to the application via our mechanism (DOUG2a , DOUG2b)



## 5. INTEGRATION PROCEDURES

This part is especially important for developers who decide to integrate their applications within the Migrating Desktop framework. Mostly the application can be integrated within the Migrating Desktop framework without any additional effort – just by using the existing functionality. Only when some specific mechanisms are required (e.g. for defining particular job input parameters, for visualising application results, etc.), do application developers have to extend the Migrating Desktop functionality by implementing appropriate plug-ins to fulfil application’s specific needs.

This chapter shows the concept of plug-ins as integration points between the Migrating Desktop and applications. The general idea of plug-ins, their architecture, interdependencies between plug-in components as well as plug-in API are described in detail. The manual for plug-in developers describing the process of creating plug-in “step-by-step” is also included.

That section contains technical details of API. Before reading that part we suggest to jump to 5.7 and get response to question “Do I need a plug-in?”.

### 5.1. GENERAL IDEA OF PLUG-INS

The idea of plug-ins was introduced to achieve two goals

- To prepare mechanism for easy adding various tools and applications;
- To make the Migrating Desktop “not so heavy” despite of the great number of handled applications and additional tools;

The Migrating Desktop plug-ins are a set of OSGi bundles with a well-defined interface designed to standardize the integration of “third party” modules and to give them easy access to resources (like user proxy, local or remote files, etc.). Every plug-in is kept in some network location and loaded only when its contents is needed. The advantages of mentioned solutions is better control that a plug-in developer has over his/her code. All changes in the plug-in code can also be introduced immediately.

### 5.2. OSGI

The OSGi stands for Open Services Gateway Initiative which evaluate to founded by Sun Microsystems, IBM, Ericsson and others in March 1999, the OSGi™ Alliance (<http://www.osgi.org/>). Among its members are more than 35 companies from quite different business areas like, for example, IBM, Nokia, Motorola, Philips, BMW, etc.

The OSGi technology is designed to provide a general-purpose, secure, and managed Java framework supporting the deployment of extensible and downloadable modules known as *bundles* that usually provide *services* - a collection of interfaces and their implementations. Bundles can be remotely installed, started, stopped, updated, or uninstalled on the fly without having to disrupt the operation of the device. The emphasis is put on designing an elegant, complete, and dynamic component model - something that is missing in standalone Java/VM environments and on life cycle management of Java packages/classes.

The original focus was on service gateways but the applicability turned out to be much wider. The OSGi specifications are now used in applications ranging from mobile phones to the open source Eclipse IDE. Other application areas include cars, industrial automation, building automation, PDAs, grid computing, entertainment (e.g. iPronto), and fleet management

As it has been mentioned above, the Migrating Desktop framework architecture is designed on the basis of the OSGi concept. The Migrating Desktop plug-ins are also designed as OSGi



bundles. Only the very basics of OSGi, necessary for implementing the Migrating Desktop plug-ins, are described in the chapter below. See pages of The OSGi Alliance (<http://www.osgi.org/>) for details.

### 5.2.1. OSGi bundles

An OSGi bundle is comprised of Java classes and other resources, which together can provide functions to end users. Bundles can share Java packages among an exporter bundle and an importer bundle in a well-defined way.

A bundle is a JAR file that:

- Contains the resources necessary to provide some functionality. These resources may be class files for the Java programming language, as well as other data such as HTML files, help files, icons, and so on. A bundle JAR file can also embed additional JAR files that are available as resources and classes. This is, however, not recursive.
- Contains a manifest file describing the contents of the JAR file and providing information about the bundle. This file uses headers to specify information needed to install correctly and activate a bundle. For example, it states dependencies on other resources, such as Java packages that must be available to the bundle before it can run.

Header name	mandatory	Description
Bundle-Name	yes	The user-friendly name of this bundle
Bundle-SymbolicName:	yes	Specifies a unique, non-localizable name for this bundle.
Bundle-Activator	no	Specifies the name of the class used to start and stop the bundle.
Bundle-Description	no	Defines a short description of this bundle.
Bundle-Vendor	no	Human-readable description of the bundle vendor
Bundle-Version	no	bundle version number
Bundle-Classpath	no	A comma-separated list of JAR file path names or directories (inside the bundle) containing classes and resources.
Export-Package	no	Declaration of exported packages
Import-Package	no	Declares the imported packages for this bundle. Imported packages must be exported by another bundle first. If an imported package cannot be found, the bundle cannot be used.
Require-Bundle	no	Specifies the required exports from another bundle.

**Fig. 5 OSGi manifest – main headers**

### 5.2.2. OSGi services

An OSGi service is a Java object instance, *registered* into an OSGi framework with a set of *properties*. Any Java object can be registered as a service, but typically it implements a well-known interface. Bundles can register services, search for them, or receive notifications when their registration state changes.

## 5.3. PLUG-INS ARCHITECTURE

### 5.3.1. Plug-in, toolkit and container

The Migrating Desktop plug-ins architecture is based on cooperation of three entities: plug-ins, toolkits and containers.



- **Plug-in** is designed to act similarly to “plug-ins” used in popular browsers– they are OSGi bundles, described by XML file and loaded “on demand” from network. This bundles provides a well-defined API that can be easily implemented and integrated with the Migrating Desktop as its “parent” application.
- **Toolkit** – interface that is implemented at the Migrating Desktop framework side and passed to the plug-in. The plug-in may use toolkit methods for gaining access to local or remote resources to use some auxiliary methods.
- **Containers** represent graphical components in which plug-ins (implementing usual java panel) are nested. Plug-ins may call a container method to perform an operation on it. The easiest case is closing the container as a response to user actions. Plug-ins may set or read some values to/from the container (e.g. plug-ins nested in the Job Submission Wizard).

### 5.3.2. Plug-in components interdependencies

The main plug-in class that implements a plug-in interface is nested within a container. The container displays javax.swing.JPanel, returned by plug-in, and calls sequentially plug-in methods. Plug-in methods may use the toolkit e.g. to access resources. Plug-ins may also call some container’s methods.

The sequence of calls of plug-in methods from the container is specific for every type of plug-in but the methods common for all plug-ins are called as follows:

- setToolContainer - passes to plug-in reference to the class that implements the container interface;
- setToolkit - passes to plug-in reference to the class that implements the toolkit interface;
- init - performs some initialization actions;
- setProperties - passes to plug-in all necessary parameters (using java.util.Properties class) specific to plug-in type;
- getPluginPanel – container gets javax.swing.JPanel for displaying;
- start - plug-in should start its execution (e.g. starts animation for visualization plug-ins);
- ... – some plug-in specific methods;
- stop - plug-in stops its execution (sequence of calls start – stop, can be called more that once in a loop);
- destroy - container calls plug-in to perform some “cleaning” actions;

## 5.4. POINTS OF INTEGRATION/TYPES OF PLUGINS

The Migrating Desktop framework can be easily extended on the basis of a set of well-defined plug-ins used for: tools integration, defining job parameters, pre-processing job parameters before submission, and visualization of job results.

### 5.4.1. Job Input Plug-in (IP1& IP2)

The Desktop provides a wizard that the user can use to specify job input details. This wizard simplifies the process of specifying parameters and limits, suggesting user default or lastly used parameters. The first panel is reserved for application-specific Job Input Plug-in.



Values of application specific parameters usually differ for each job and for such case specific Job input plug-in should be prepared.

For parameters that values are changed very rarely we suggest to keep parameters in files, and attach file as a parameter to job. (Especially if there is huge number of such parameters)

If the application needs some *specific parameters*, they can be defined in two ways:

- using *plug-in* implemented by application developers and displayed as the content of the first panel of the Job Submission Wizard. (IP1)
- Using a generic XML plug-in that interprets the argument description given in XML to a graphical form. (IP2)

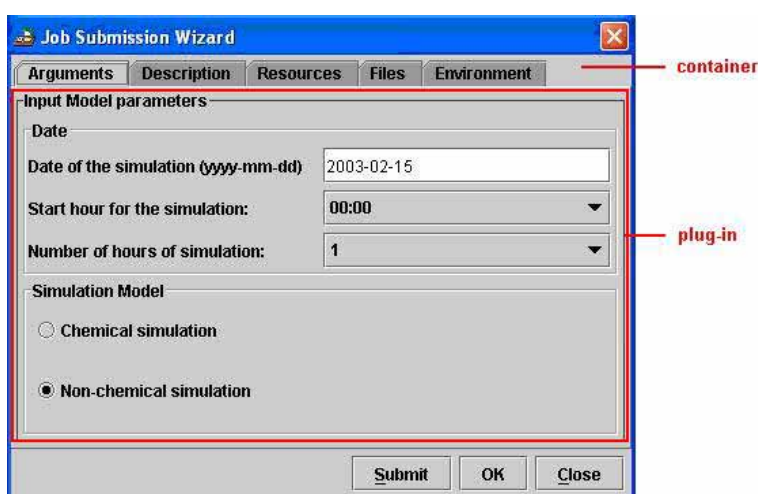


Fig. 6 Job Submission Wizard – example of plug-in for defining specific application parameters

#### 5.4.2. Job Input Plug-in API

**public void setPluginContainer(JobPluginContainer container);**

Sets plug-in container - graphical component in which plug-in is nested

Parameters:

- container – plug-in container

**public void setPluginToolkit(JobPluginToolkit toolkit);**

Sets plug-in toolkit - auxiliary class that can be used for accessing resources, etc.

Parameters:

- toolkit - plug-in toolkit

**public void setProperties(Properties props);**

Sets plug-in properties (some initialisation values depending on plug-in type).

Parameters:

- props - plug-in properties

**public void init() throws PluginException;**

Initialize plug-in. Called by container to let plug-in know that it should perform some initialization actions. It may be an empty method.

Throws:

- *PluginException* - in case of any errors.



**public void start() throws PluginException;**

Start plug-in execution. Called by container to let plug-in know that it has been added to container, it is now visible and should start its activities.

Throws:

- *PluginException* - in case of any errors.

**public void stop() throws PluginException;**

Stop plug-in execution. Called by container to let plug-in know that it is not longer visible and will be removed from container. It may be an empty method.

Throws:

- *PluginException* - in case of any errors.

**public void destroy();**

Destroy plug-in. Called container to let plug-in know that it will be no longer used and should perform some "cleanup" operation. It may be an empty method

**public JPanel getPluginPanel() throws PluginException;**

Method gets plug-in graphic to put it within container. (Plug-in should use container window as its main window).

Return:

- *JPanel* main plug-in panel

Throws:

- *PluginException* - in case of any errors.

**public boolean onSubmit() throws PluginException;**

Called just before submitting a job. If some data or scripts should be prepared, they should be created here. This function can also check if parameters are set correctly. If return is false, job will be not submitted.

Return:

- *boolean* - **true** indicates that job can be submitted, **false** - otherwise

Throws:

- *PluginException* - in case of any errors.

**public Argument[] getArguments() throws PluginException;**

Gets job specific arguments - used to save the job and to create a command line to start a job.

Return:

- *Argument[]* - array of arguments objects

Throws:

- *PluginException* - in case of any errors.

**public void setArguments(Argument[] arg);**

Sets application arguments that are presented in the Job Submission Wizard.

Parameters:

- *arg* - array of arguments

### 5.4.3. Job Input Plug-in Toolkit API

**GSSCredential getUserCredential() throws PluginToolkitException;**

Gets credential of Migrating Desktop user.

Return:

- *GSSCredential* - user credential

Throws:

- *PluginToolkitException* - in case of any errors.

**InputStream getInputStream(String fileld, int transferMode) throws PluginToolkitException;**

Gets input stream for given file.



Parameters:

- *fileId* - file unique identifier
- *transferMode* - determines ascii or binary transfer (defined in PluginConstants class)

Return:

- *InputStream* - input stream created for specified file

Throws:

- *PluginToolkitException* - in case of any errors.

**byte[] readFile(String fileId, int transferMode) throws PluginToolkitException;**

Reads file and returns it as byte array. This method (due to memory limitation) shall be used for small files only. The method *getInputStream* is recommended for reading huge files.

Parameters:

- *fileId* - file unique identifier
- *transferMode* - determines ascii or binary transfer (defined in PluginConstants class)

Return:

- *byte[]* - read file

Throws:

- *PluginToolkitException* - in case of any errors.

**String getPhysicalLocation(String fileId) throws PluginToolkitException;**

Gets physical location (URL) of file of given identifier

Parameters:

- *fileId* - file unique identifier

Return:

- *String* - physical location of file (URL)

Throws:

- *PluginToolkitException* - in case of any errors.

**ImageIcon loadImageIcon(String url) throws PluginToolkitException;**

Loads image icon from the URL specified

Parameters:

- *url* - URL of image

Return:

- *ImageIcon* - loaded image

Throws:

- *PluginToolkitException* - in case of any errors.

**void savePluginProperties(String pluginId, Properties props) throws PluginToolkitException;**

Saves "developer defined" properties for plug-in

Parameters:

- *pluginId* - plug-in unique id
- *props* - plug-in properties

Throws:

- *PluginToolkitException* - in case of any errors.

**Properties loadPluginProperties(String pluginId) throws PluginToolkitException;**

Loads "developer defined" properties for plug-in.

Parameters:

- *pluginId* - plug-in unique identifier

Return:

- *Properties* - plug-in properties

Throws:



- *PluginToolkitException* - in case of any errors.

**NodeInfo browseFileSystem(String startDirId, int selectionMode) throws PluginToolkitException;**

Opens the Migrating Desktop file chooser (so called Grid Explorer) in selected mode and returns identifier of file or dir that user chose; File chooser starts in directory specified by startDirId parameter.

Parameters:

- *startDirId* - initial directory;
- *selectionMode* - "file" or "directory" selection mode;

Return:

- *NodeInfo* – structure describing file or dir selected by user

Throws:

- *PluginToolkitException* - in case of any errors.

**NodeInfo[] listDirectory(String startDirId) throws PluginToolkitException;**

Opens the Migrating Desktop file chooser (so called Grid Explorer) in "directory selection" mode and returns array of file identifier belonging to directory that user choose; File chooser starts in directory specified by startDirId parameter.

Parameters:

- *startDirId* - initial directory;

Return:

- *NodeInfo[]* – array of structures described files and directories belonging to selected directory

Throws:

- *PluginToolkitException* - in case of any errors.

**OutputStream getOutputStream(String fileId, boolean append, int transferMode) throws PluginToolkitException;**

Gets output stream for given file for writing

Parameters:

- *fileId* - file unique identifier
- *append* - determines if content of file is overwritten
- *transferMode* - determines ascii or binary transfer (defined in PluginConstants class)

Return:

- *OutputStream* - stream created for specified file

Throws:

- *PluginToolkitException* - in case of any errors.

**void writeFile(String fileId, boolean append, int transferMode, byte buffer[]) throws PluginToolkitException;**

Writes file from byte array. This method (due to memory limitations) shall be used for small files only. The method getOutputStream is recommended for writing huge files.

Parameters:

- *fileId* - file unique identifier
- *append* - determines if content of file is overwritten
- *transferMode* - determines ascii or binary transfer (defined in PluginConstants class)
- *buffer* - content of file to be written

Throws:

- *PluginToolkitException* - in case of any errors.

**void writeTempFile(String name, byte buffer[]) throws PluginToolkitException;**



Writes file from byte array in temporary directory. This method (due to memory limitations) shall be used for small files only. The method `getOutputStream` is recommended for writing huge files.

Parameters:

- *name* - file name
- *buffer* - content of file to be written

Throws:

- *PluginToolkitException* - in case of any errors.

#### 5.4.4. Parameters passed to job viewer plug-in

General parameters.

Property name	Description
Name	Job name set by the user
Status	Job status (in format "number:string")
JobId	Unique job identifier (this does not need to be EDG JobId or Globus JobId)
EndDate	Job finishing date as returned by RB
User	Distinguish name of submitter (certificate subject)
SubmissionDate	Date of job submission
StartDate	Date when job was starter
Host	Hostname on which job was run
Type	Grid type.
Application	Application identifier

#### Arguments

For every argument:

Property name	Description
<code>Argument.&lt;arg_name&gt;.Value</code>	Value of an argument
<code>Argument.&lt;arg_name&gt;.Prefix</code>	Prefix of the argument as it should appear in command line (eg. -f=)
<code>Argument.&lt;arg_name&gt;.CommandLine</code>	"true" or "false" indicates whether this argument should be added to command line (in a form Prefix+Value eg. -f=5)  Possible use of this field is when one command line argument is generated from several graphical components. For example, application plugin can show three input boxes and ask about RGB color components. Then arguments with names Rcolor, Gcolor and Bcolor should have CommandLine flag set to "false" and argument RGBcolor with value computed from RGB components should have CommandLine flag set to "true" and will be passed to command line.

#### Resources



For every resource:

Property name	Description
Resource.<res_name>.Value	Name and Value of job resource requirements. Currently the name can be one of: "cpucount", "mincpu", "maxcpu", "maxmem", "minmem", "maxtime", "mintime", "hostname", "ostype", "osname", "osversion", "cpuspeed", "application", "jobtype"

## Files

For every file:

Property name	Description
File.<file_log_name>.Id	Virtual Directory unique file identifier
File.<file_log_name>.Type	"in", "out", "inout", "refr", "temp", "StdInput", "StdOutput", "StdError";
File.<file_log_name>.RefreshTime	Time (in seconds) after which "visualizer" should reload file

## Environment

For every environment variable:

Property name	Description
Variable.<var_name>.Value	Environment variable (like PATH, USER_HOME, etc)

### 5.4.5. XML schema description of job specific input parameters

To make defining specific job input parameters even easier, the application developer may utilize "ready-to-use" plug-in that can interpret an XML schema description for automatic creation of the Job Submission Wizard's Argument panel content.

The XML schema specification is a subset of tags and roles which have been defined and described in documents of the World Wide Web Consortium (W3C <http://www.w3c.org/XML/Schema>). This paragraph describes the specification of XML schema recognized and interpreted by the XML parser implemented in the Migrating Desktop framework. It is used to define application parameters in XML schema files and to represent them as a set of tabs, graphic controls and relations between them in the Job Submission Wizard's Argument panel.

### XML schema tags describing Java GUI controls

#### ➤ JTextField

It can be used to collect application parameters as simple strings entered by the user. The XML code example describing a single JTextField control is the following:

```
<xs:element name="str1">  
  <xs:annotation>  
    <xs:appinfo>String</xs:appinfo>  
    <xs:documentation>Type some text here!</xs:documentation>  
  </xs:annotation>  
</xs:element>
```



```
</xs:annotation>
<xs:complexType>
  <xs:attribute fixed="-s" name="prefix" type="xs:string" use="required"/>
  <xs:attribute fixed="100" name="width" type="xs:integer" use="optional"/>
</xs:complexType>
</xs:element>
```

#### ➤ JCheckBox

It can be used to collect application parameters with the boolean value. If JCheckBox is checked (has the “true” value), its prefix string will appear in the application command line. Otherwise it will not appear. The XML code example describing a single JCheckBox control is the following:

```
<xs:element name="Bool">
  <xs:annotation>
    <xs:appinfo>CheckBox</xs:appinfo>
    <xs:documentation>Select or unselect this checkbox!</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:boolean">
        <xs:attribute name="width" fixed="100"/>
        <xs:attribute name="prefix" fixed="-b"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

#### ➤ JComboBox

It can be used to collect application parameters from a set of predefined string values. If the JComboBox control does not need to have predefined width in the GUI panel and separate prefix in the application command line, it can be coded as below:

```
<xs:element name="Box2">
  <xs:annotation>
    <xs:appinfo>simple ComboBox</xs:appinfo>
    <xs:documentation>Select another element!</xs:documentation>
  </xs:annotation>
  <xs:simpleType name="MyComboType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="_value1"/>
      <xs:enumeration value="_value2"/>
      <xs:enumeration value="_value3"/>
      <xs:enumeration value="_value4"/>
      <xs:enumeration value="_value5"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

If a more complex structure of JComboBox is required, it should be coded in two parts. The first one is the “xs:simpleType” tag describing allowed values (a set of values) and is placed directly as a child of the “xs:schema” tag at the beginning of the document. The second part is the “xs:element” tag which extends the previously defined “simpleType” with “width” and “prefix” attributes. The XML code example describing a single JComboBox control is the following:

```
<xs:simpleType name="MyComboType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="value1"/>
```



```
<xs:enumeration value="value2"/>
<xs:enumeration value="value3"/>
<xs:enumeration value="value4"/>
<xs:enumeration value="value5"/>
</xs:restriction>
<xs:element name="Box">
  <xs:annotation>
    <xs:appinfo>ComboBox</xs:appinfo>
    <xs:documentation>Select one element!</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="MyComboType">
        <xs:attribute name="width" fixed="100"/>
        <xs:attribute name="prefix" fixed="-combo"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
</xs:simpleType>
```

#### ➤ JSlider

It can be used to collect application parameters from a range of predefined integer values. The XML code example describing a single JSlider control is the following:

```
<xs:element name="range">
  <xs:annotation>
    <xs:appinfo>JSlider</xs:appinfo>
    <xs:documentation>Value between 'X' and 'Y'</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:integer">
        <xs:attribute fixed="-slider" name="prefix"/>
        <xs:attribute fixed="100" name="width"/>
        <xs:attribute name="rangeMin" fixed="0"/>
        <xs:attribute name="rangeMax" fixed="7"/>
        <xs:attribute name="rangeStep" fixed="1"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

### XML schema structure recognized by the Migrating Desktop

#### ➤ Default value

Each “xs:element” tag defining GUI control can also define the default value for this element. This value is set on the first display of the application parameters panel. The default value can be coded as the following:

```
<xs:element name="MyElement" default="2">
  ...
</xs:element>
```

#### ➤ Prefix in application command line



The prefix which should be placed before the given element value in the application command line can be coded as the following:

```
<xs:element name="range">
....
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:integer">
        <xs:attribute fixed="-slider" name="prefix"/>
        ...
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

This attribute (unchanged with no additional strings) will be placed in the Commandline for the application, so it should also contain "" character if required. An empty string means no prefix will be used for a given parameter.

➤ Parameter label

Each GUI control can have its own label (placed on the left side) which informs the user about the meaning of this application parameter. Such label should be coded as the following:

```
<xs:element name="range">
  <xs:annotation>
    <xs:appinfo>My Range</xs:appinfo>
    ...
  </xs:annotation>
  ...
</xs:element>
```

➤ Parameter description

Each GUI control can have its own description (placed below) which describes the meaning of this application parameter. Such description should be coded as the following:

```
<xs:element name="range">
  <xs:annotation>
    ...
    <xs:documentation>Value between 'X' and 'Y'</xs:documentation>
  </xs:annotation>
  ...
</xs:element>
```

➤ Predefined GUI control width

The width of the GUI control can be set by an additional attribute as the following:

```
<xs:element name="range">
...
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:integer">
        <xs:attribute name="width" fixed="100"/>
        ...
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```



## Dynamic GUI in XML schema structure

In the XML schema structure recognized by the Migrating Desktop parser a kind of dynamic GUI can be implemented. The value of one parameter can decide about the quantity of others. Elements are related with each other by an additional “reference” attribute. An element whose quantity can be dynamically changed contains two additional attributes “minOccurs” and “maxOccurs” in its definition. These attributes define a range of instances of this element. minOccurs == 0 means this element is optional and may not exist in GUI representation.

The example below shows 4 elements with names: “Box2”, “set”, “Second\_TAB” and “dyn”. “Box2” element is represented as JComboBox with a default value “2” this element has attribute “reference” (with value “Second\_TAB”, which is a name of the referenced element), so it decides about the quantity of the “Second\_TAB” element. The “Second\_TAB” element (referenced by “Box2”) contains more sub-elements

so it is represented as a separate TAB with a label “Second\_TAB”. This element is optional (minOccurs == 0) so it may not exist in GUI when the user sets the “Box2” value to 0. There can be max 10 (maxOccurs == 10) “Second\_TAB” elements. The “set” element is represented as simple JTextField and contains “reference” attribute points to the “dyn” element, so the value of the “set” element decides about the quantity of “dyn” elements. The “dyn” element (referenced by “set”) is a simple JTextField control with label “dyn”. This element is optional (minOccurs == 0) so it may not exist in GUI when the user

sets the “set” value to 0. There can be max 10 (maxOccurs == 10) “dyn” elements. The quantity of the “dyn” element (defined by the “set” value) is set **on each** “Second\_TAB” tab.

```
<xs:element name="Box2" default="2">
  <xs:annotation>
    <xs:appinfo>Nr of second tabs</xs:appinfo>
    <xs:documentation>Select number of tabs!</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="TABS">
        <xs:attribute name="width" fixed="100"/>
        <xs:attribute name="prefix" fixed="-tabs"/>
        <xs:attribute name="reference" fixed="Second_TAB"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
<xs:element name="set">
  <xs:annotation>
    <xs:appinfo>Number of layers</xs:appinfo>
    <xs:documentation>Sets a number of controls on secondtab!</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:attribute name="reference" fixed="dyn"/>
    <xs:attribute name="width" fixed="100"/>
  </xs:complexType>
</xs:element>
<xs:element name="Second_TAB" minOccurs="0" maxOccurs="10">
  <xs:annotation>
    <xs:documentation>Another example of TAB</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="dyn" minOccurs="0" maxOccurs="10"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```



```
</xs:sequence>  
</xs:complexType>  
</xs:element>
```

#### 5.4.6. Job Process Plug-in (IP3)

In some cases the job description built by the Job Submission Wizard needs to be pre-processed (e.g. to enable monitoring, profiling, etc). The application developer can define a tool that changes the job description before submission and adds it to the Migrating Desktop framework using the concept of plug-ins.

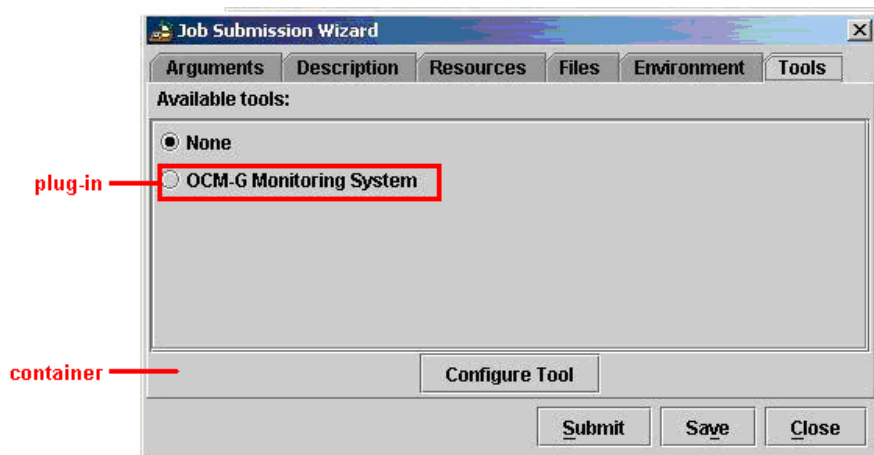


Fig. 7 Job Submission Wizard – pre-processing tools

#### 5.4.7. Job Process Plug-in API

##### **public void setPluginContainer(JobPluginContainer container);**

Sets plug-in container - graphical component in which plug-in is nested

Parameters:

- container – plug-in container

##### **public void setPluginToolkit(JobPluginToolkit toolkit);**

Sets plug-in toolkit - auxiliary class that can be used for accessing resources, etc.

Parameters:

- toolkit - plug-in toolkit

##### **public void setProperties(Properties props);**

Sets plug-in properties (some initialisation values depending on plug-in type).

Parameters:

- props - plug-in properties

##### **public void init() throws PluginException;**

Initialize plug-in. Called by container to let plug-in know that it should perform some initialization actions. It may be an empty method.

Throws:

- *PluginException* - in case of any errors.

##### **public void start() throws PluginException;**

Start plug-in execution. Called by container to let plug-in know that it has been added to container, it is now visible and should start its activities.

Throws:



- *PluginException* - in case of any errors.

**public void stop() throws PluginException;**

Stop plug-in execution. Called by container to let plug-in know that it is not longer visible and will be removed from container. It may be an empty method.

Throws:

- *PluginException* - in case of any errors.

**public void destroy();**

Destroy plug-in. Called container to let plug-in know that it will be no longer used and should perform some "cleanup" operation. It may be an empty method

**public JPanel getPluginPanel() throws PluginException;**

Method gets plug-in graphic to put it within container. (Plug-in should use container window as its main window).

Return:

- *JPanel* main plug-in panel

Throws:

- *PluginException* - in case of any errors.

**public boolean onSubmit() throws PluginException;**

Called just before submitting a job. If some data or scripts should be prepared, they should be created here. This function can also check if parameters are set correctly. If return is false, job will be not submitted.

Return:

- *boolean* - **true** indicates that job can be submitted, **false** - otherwise

Throws:

- *PluginException* - in case of any errors.

**public Argument[] getArguments() throws PluginException;**

Gets job specific arguments - used to save the job and to create a command line to start a job.

Return:

- *Argument[]* - array of arguments objects

Throws:

- *PluginException* - in case of any errors.

**public void setArguments(Argument[] arg);**

Sets application arguments that are presented in the Job Submission Wizard.

Parameters:

- *arg* - array of arguments

**public void configure() throws PluginException;**

Called by job submission wizard when user clicks the button for tool configuration. Can show user a dialog for parameter configuration.

Throws:

- *PluginException* - in case of any errors.

#### **5.4.8. Job Process Plug-in Toolkit API**

**GSSCredential getUserCredential() throws PluginToolkitException;**

Gets credential of Migrating Desktop user.

Return:

- *GSSCredential* - user credential

Throws:

- *PluginToolkitException* - in case of any errors.

**InputStream getInputStream(String fileId, int transferMode) throws PluginToolkitException;**



Gets input stream for given file.

Parameters:

- *fileId* - file unique identifier
- *transferMode* - determines ascii or binary transfer (defined in PluginConstants class)

Return:

- *InputStream* - input stream created for specified file

Throws:

- *PluginToolkitException* - in case of any errors.

**byte[] readFile(String fileId, int transferMode) throws PluginToolkitException;**

Reads file and returns it as byte array. This method (due to memory limitation) shall be used for small files only. The method *getInputStream* is recommended for reading huge files.

Parameters:

- *fileId* - file unique identifier
- *transferMode* - determines ascii or binary transfer (defined in PluginConstants class)

Return:

- *byte[]* - read file

Throws:

- *PluginToolkitException* - in case of any errors.

**String getPhysicalLocation(String fileId) throws PluginToolkitException;**

Gets physical location (URL) of file of given identifier

Parameters:

- *fileId* - file unique identifier

Return:

- *String* - physical location of file (URL)

Throws:

- *PluginToolkitException* - in case of any errors.

**ImageIcon loadImageIcon(String url) throws PluginToolkitException;**

Loads image icon from the URL specified

Parameters:

- *url* - URL of image

Return:

- *ImageIcon* - loaded image

Throws:

- *PluginToolkitException* - in case of any errors.

**void savePluginProperties(String pluginId, Properties props) throws PluginToolkitException;**

Saves "developer defined" properties for plug-in

Parameters:

- *pluginId* - plug-in unique id
- *props* - plug-in properties

Throws:

- *PluginToolkitException* - in case of any errors.

**Properties loadPluginProperties(String pluginId) throws PluginToolkitException;**

Loads "developer defined" properties for plug-in.

Parameters:

- *pluginId* - plug-in unique identifier

Return:

- *Properties* - plug-in properties



Throws:

- *PluginToolkitException* - in case of any errors.

**NodeInfo browseFileSystem(String startDirId, int selectionMode) throws PluginToolkitException;**

Opens Migrating Desktop file chooser (so called Grid Explorer) in selected mode and returns identifier of file or dir that user chose; File chooser starts in directory specified by startDirId parameter.

Parameters:

- *startDirId* - initial directory;
- *selectionMode* - "file" or "directory" selection mode;

Return:

- *NodeInfo* – structure describing file or dir selected by user

Throws:

- *PluginToolkitException* - in case of any errors.

**NodeInfo[] listDirectory(String startDirId) throws PluginToolkitException;**

Opens Migrating Desktop file chooser (so called Grid Explorer) in “directory selection” mode and returns array of file identifier belonging to directory that user choose; File chooser starts in directory specified by startDirId parameter.

Parameters:

- *startDirId* - initial directory;

Return:

- *NodeInfo[]* – array of structures described files and directories belonging to selected directory

Throws:

- *PluginToolkitException* - in case of any errors.

**OutputStream getOutputStream(String fileId, boolean append, int transferMode) throws PluginToolkitException;**

Gets output stream for given file for writing

Parameters:

- *fileId* - file unique identifier
- *append* - determines if content of file is overwritten
- *transferMode* - determines ascii or binary transfer (defined in PluginConstants class)

Return:

- *OutputStream* - stream created for specified file

Throws:

- *PluginToolkitException* - in case of any errors.

**void writeFile(String fileId, boolean append, int transferMode, byte buffer[]) throws PluginToolkitException;**

Writes file from byte array. This method (due to memory limitations) shall be used for small files only. The method *getOutputStream* is recommended for writing huge files.

Parameters:

- *fileId* - file unique identifier
- *append* - determines if content of file is overwritten
- *transferMode* - determines ascii or binary transfer (defined in PluginConstants class)
- *buffer* - content of file to be written

Throws:

- *PluginToolkitException* - in case of any errors.

**void writeTempFile(String name, byte buffer[]) throws PluginToolkitException;**



Writes file from byte array in temporary directory. This method (due to memory limitations) shall be used for small files only. The method `getOutputStream` is recommended for writing huge files.

Parameters:

- *name* - file name
- *buffer* - content of file to be written

Throws:

- *PluginToolkitException* - in case of any errors.

#### 5.4.9. File Viewer Plug-in (IP4)

The Migrating Desktop framework has a built-in set of viewers for standard format files – bitmaps (\*.bmp), files containing images (\*.png, \*.jpg, \*.gif) or text (\*.txt). This set can be extended by File Viewers Plug-ins used for visualising other formats of files, if necessary.



Fig. 8 Image file viewer

#### 5.4.10. File Viewer Plug-in API

**public void setPluginContainer(FileViewerPluginContainer container);**

Sets plug-in container - graphical component in which plug-in is nested

Parameters:

- container – plug-in container

**public void setPluginToolkit(FileViewerPluginToolkit toolkit);**

Sets plug-in toolkit - auxiliary class that can be used for accessing resources, etc.

Parameters:

- toolkit - plug-in toolkit

**public void setProperties(Properties props);**

Sets plug-in properties (some initialisation values depending on plug-in type).

Parameters:

- props - plug-in properties

**public void init() throws PluginException;**

Initialize plug-in. Called by container to let plug-in know that it should perform some initialization actions. It may be an empty method.



Throws:

- *PluginException* - in case of any errors.

**public void start() throws PluginException;**

Start plug-in execution. Called by container to let plug-in know that it has been added to container, it is now visible and should start its activities.

Throws:

- *PluginException* - in case of any errors.

**public void stop() throws PluginException;**

Stop plug-in execution. Called by container to let plug-in know that it is not longer visible and will be removed from container. It may be an empty method.

Throws:

- *PluginException* - in case of any errors.

**public void destroy();**

Destroy plug-in. Called container to let plug-in know that it will be no longer used and should perform some "cleanup" operation. It may be an empty method

**public JPanel getPluginPanel() throws PluginException;**

Method gets plug-in graphic to put it within container. (Plug-in should use container window as its main window).

Return:

- *JPanel* main plug-in panel

Throws:

- *PluginException* - in case of any errors.

**public void setFileId(String fileId);**

Sets identifier of visualised file

Parameters:

- *fileID* - file identifier

#### 5.4.11. File Viewer Toolkit API

**GSSCredential getUserCredential() throws PluginToolkitException;**

Gets credential of the Migrating Desktop user.

Return:

- *GSSCredential* - user credential

Throws:

- *PluginToolkitException* - in case of any errors.

**InputStream getInputStream(String fileId, int transferMode) throws PluginToolkitException;**

Gets input stream for given file.

Parameters:

- *fileId* - file unique identifier
- *transferMode* - determines ascii or binary transfer (defined in *PluginConstants* class)

Return:

- *InputStream* - input stream created for specified file

Throws:

- *PluginToolkitException* - in case of any errors.

**byte[] readFile(String fileId, int transferMode) throws PluginToolkitException;**

Reads file and returns it as byte array. This method (due to memory limitation) shall be used for small files only. The method *getInputStream* is recommended for reading huge files.



Parameters:

- *fileId* - file unique identifier
- *transferMode* - determines ascii or binary transfer (defined in *PluginConstants* class)

Return:

- *byte[]* - read file

Throws:

- *PluginToolkitException* - in case of any errors.

**String getPhysicalLocation(String fileId) throws PluginToolkitException;**

Gets physical location (URL) of file of given identifier

Parameters:

- *fileId* - file unique identifier

Return:

- *String* - physical location of file (URL)

Throws:

- *PluginToolkitException* - in case of any errors.

**ImageIcon loadImageIcon(String url) throws PluginToolkitException;**

Loads image icon from the URL specified

Parameters:

- *url* - URL of image

Return:

- *ImageIcon* - loaded image

Throws:

- *PluginToolkitException* - in case of any errors.

**void savePluginProperties(String pluginId, Properties props) throws PluginToolkitException;**

Saves "developer defined" properties for plug-in

Parameters:

- *pluginId* - plug-in unique id
- *props* - plug-in properties

Throws:

- *PluginToolkitException* - in case of any errors.

**Properties loadPluginProperties(String pluginId) throws PluginToolkitException;**

Loads "developer defined" properties for plug-in.

Parameters:

- *pluginId* - plug-in unique identifier

Return:

- *Properties* - plug-in properties

Throws:

- *PluginToolkitException* - in case of any errors.



#### 5.4.12. Job Viewer Plug-in (IP5 & IP6)

When the output that the job produces is a standard file (as e.g. txt, jpg, bmp), it can be visualized using the Migrating Desktop built-in file viewers. Some applications require a more sophisticated graphical visualisation of job results (e.g. animation). In the Migrating Desktop every type of application can be related to its “visualisator” - also defined as the Migrating Desktop plug-in.

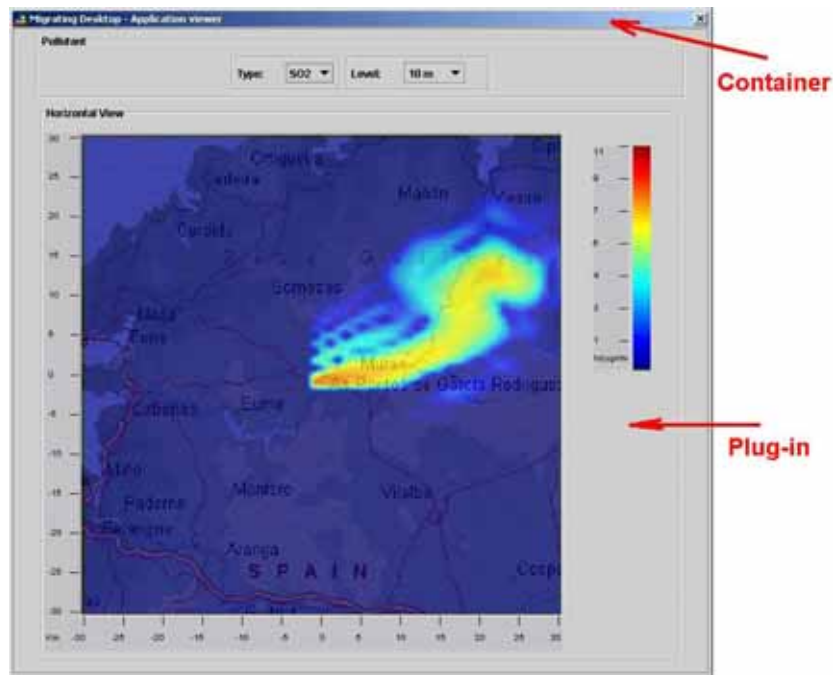


Fig. 9 Visualisation of air pollution application

#### 5.4.13. Job Viewer Plug-in API

**public void setPluginContainer(JobViewerPluginContainer container);**

Sets plug-in container - graphical component in which plug-in is nested

Parameters:

- container – plug-in container

**public void setPluginToolkit(JobViewerPluginToolkit toolkit);**

Sets plug-in toolkit - auxiliary class that can be used for accessing resources, etc.

Parameters:

- toolkit - plug-in toolkit

**public void setProperties(Properties props);**

Sets plug-in properties (some initialisation values depending on plug-in type).

Parameters:

- props - plug-in properties

**public void init() throws PluginException;**

Initialize plug-in. Called by container to let plug-in know that it should perform some initialization actions. It may be an empty method.

Throws:

- *PluginException* - in case of any errors.



**public void start() throws PluginException;**

Start plug-in execution. Called by container to let plug-in know that it has been added to container, it is now visible and should start its activities.

Throws:

- *PluginException* - in case of any errors.

**public void stop() throws PluginException;**

Stop plug-in execution. Called by container to let plug-in know that it is not longer visible and will be removed from container. It may be an empty method.

Throws:

- *PluginException* - in case of any errors.

**public void destroy();**

Destroy plug-in. Called container to let plug-in know that it will be no longer used and should perform some "cleanup" operation. It may be an empty method

**public JPanel getPluginPanel() throws PluginException;**

Method gets plug-in graphic to put it within container. (Plug-in should use container window as its main window).

Return:

- *JPanel* main plug-in panel

Throws:

- *PluginException* - in case of any errors.

**public void setJobId(String jobId);**

Sets identifier of visualized job

Parameters:

- *jobID* - file identifier

#### **5.4.14. Job Viewer Toolkit API**

**GSSCredential getUserCredential() throws PluginToolkitException;**

Gets credential of Migrating Desktop user.

Return:

- *GSSCredential* - user credential

Throws:

- *PluginToolkitException* - in case of any errors.

**InputStream getInputStream(String fileId, int transferMode) throws PluginToolkitException;**

Gets input stream for given file.

Parameters:

- *fileId* - file unique identifier
- *transferMode* - determines ascii or binary transfer (defined in PluginConstants class)

Return:

- *InputStream* - input stream created for specified file

Throws:

- *PluginToolkitException* - in case of any errors.

**byte[] readFile(String fileId, int transferMode) throws PluginToolkitException;**

Reads file and returns it as byte array. This method (due to memory limitation) shall be used for small files only. The method *getInputStream* is recommended for reading huge files.

Parameters:

- *fileId* - file unique identifier
- *transferMode* - determines ascii or binary transfer (defined in PluginConstants class)



Return:

- *byte[]* - read file

Throws:

- *PluginToolkitException* - in case of any errors.

**String getPhysicalLocation(String fileId) throws PluginToolkitException;**

Gets physical location (URL) of file of given identifier

Parameters:

- *fileId* - file unique identifier

Return:

- *String* - physical location of file (URL)

Throws:

- *PluginToolkitException* - in case of any errors.

**ImageIcon loadImageIcon(String url) throws PluginToolkitException;**

Loads image icon from the URL specified

Parameters:

- *url* - URL of image

Return:

- *ImageIcon* - loaded image

Throws:

- *PluginToolkitException* - in case of any errors.

**void savePluginProperties(String pluginId, Properties props) throws PluginToolkitException;**

Saves "developer defined" properties for plug-in

Parameters:

- *pluginId* - plug-in unique id
- *props* - plug-in properties

Throws:

- *PluginToolkitException* - in case of any errors.

**Properties loadPluginProperties(String pluginId) throws PluginToolkitException;**

Loads "developer defined" properties for plug-in.

Parameters:

- *pluginId* - plug-in unique identifier

Return:

- *Properties* - plug-in properties

Throws:

- *PluginToolkitException* - in case of any errors.

**public InputStream[] getInteractiveInputStreams(String jobId) throws PluginToolkitException;**

Gets input streams for interactive job of chosen identifier

Parameters:

- *jobId* - job identifier

Return:

- *InputStream[]* - interactive input streams

Throws:

- *PluginToolkitException* - in case of any errors.

**public OutputStream getInteractiveOutputStream(String jobId) throws PluginToolkitException;**

Parameters:

- *jobId* - job identifier



Return:

- *OutputStream* - interactive output stream

Throws:

- *PluginToolkitException* - in case of any errors.

#### 5.4.15. Tool Plug-in

Another example of MD plug-ins is Tool Plug-in. Using this kind of plug-in a developer can integrate any type of Java applet or application within the Migrating Desktop framework. Similar to other plug-ins, tools are downloaded from different network localisations on demand.



Fig. 10 GridBenchmark – example of tool integrated within Migrating Desktop

#### 5.4.16. Tool Plug-in API

**public void setPluginContainer(ToolPluginContainer container);**

Sets plug-in container - graphical component in which plug-in is nested

Parameters:

- container – plug-in container

**public void setPluginToolkit(ToolPluginToolkit toolkit);**

Sets plug-in toolkit - auxiliary class that can be used for accessing resources, etc.

Parameters:

- toolkit - plug-in toolkit

**public void setProperties(Properties props);**

Sets plug-in properties (some initialisation values depending on plug-in type).

Parameters:

- props - plug-in properties



**public void init() throws PluginException;**

Initialize plug-in. Called by container to let plug-in know that it should perform some initialization actions. It may be an empty method.

Throws:

- *PluginException* - in case of any errors.

**public void start() throws PluginException;**

Start plug-in execution. Called by container to let plug-in know that it has been added to container, it is now visible and should start its activities.

Throws:

- *PluginException* - in case of any errors.

**public void stop() throws PluginException;**

Stop plug-in execution. Called by container to let plug-in know that it is not longer visible and will be removed from container. It may be an empty method.

Throws:

- *PluginException* - in case of any errors.

**public void destroy();**

Destroy plug-in. Called container to let plug-in know that it will be no longer used and should perform some "cleanup" operation. It may be an empty method

**public JPanel getPluginPanel() throws PluginException;**

Method gets plug-in graphic to put it within container. (Plug-in should use container window as its main window).

Return:

- *JPanel* main plug-in panel

Throws:

- *PluginException* - in case of any errors.

### 5.4.17. Tool Plug-in Toolkit API

**GSSCredential getUserCredential() throws PluginToolkitException;**

Gets credential of Migrating Desktop user.

Return:

- *GSSCredential* - user credential

Throws:

- *PluginToolkitException* - in case of any errors.

**InputStream getInputStream(String fileId, int transferMode) throws PluginToolkitException;**

Gets input stream for given file.

Parameters:

- *fileId* - file unique identifier
- *transferMode* - determines ascii or binary transfer (defined in PluginConstants class)

Return:

- *InputStream* - input stream created for specified file

Throws:

- *PluginToolkitException* - in case of any errors.

**byte[] readFile(String fileId, int transferMode) throws PluginToolkitException;**

Reads file and returns it as byte array. This method (due to memory limitation) shall be used for small files only. The method *getInputStream* is recommended for reading huge files.

Parameters:

- *fileId* - file unique identifier
- *transferMode* - determines ascii or binary transfer (defined in PluginConstants class)



Return:

- *byte[]* - read file

Throws:

- *PluginToolkitException* - in case of any errors.

**String getPhysicalLocation(String fileId) throws PluginToolkitException;**

Gets physical location (URL) of file of given identifier

Parameters:

- *fileId* - file unique identifier

Return:

- *String* - physical location of file (URL)

Throws:

- *PluginToolkitException* - in case of any errors.

**ImageIcon loadImageIcon(String url) throws PluginToolkitException;**

Loads image icon from the URL specified

Parameters:

- *url* - URL of image

Return:

- *ImageIcon* - loaded image

Throws:

- *PluginToolkitException* - in case of any errors.

**void savePluginProperties(String pluginId, Properties props) throws PluginToolkitException;**

Saves "developer defined" properties for plug-in

Parameters:

- *pluginId* - plug-in unique id
- *props* - plug-in properties

Throws:

- *PluginToolkitException* - in case of any errors.

**Properties loadPluginProperties(String pluginId) throws PluginToolkitException;**

Loads "developer defined" properties for plug-in.

Parameters:

- *pluginId* - plug-in unique identifier

Return:

- *Properties* - plug-in properties

Throws:

- *PluginToolkitException* - in case of any errors.

**NodeInfo browseFileSystem(String startDirId, int selectionMode) throws PluginToolkitException;**

Opens Migrating Desktop file chooser (so called Grid Explorer) in selected mode and returns identifier of file or dir that user chose; File chooser starts in directory specified by startDirId parameter.

Parameters:

- *startDirId* - initial directory;
- *selectionMode* - "file" or "directory" selection mode;

Return:

- *NodeInfo* – structure describing file or dir selected by user

Throws:

- *PluginToolkitException* - in case of any errors.



**NodeInfo[] listDirectory(String startDirId) throws PluginToolkitException;**

Opens Migrating Desktop file chooser (so called Grid Explorer) in “directory selection” mode and returns array of file identifier belonging to directory that user choose; File chooser starts in directory specified by startDirId parameter.

Parameters:

- *startDirId* - initial directory;

Return:

- *NodeInfo[]* – array of structures described files and directories belonging to selected directory

Throws:

- *PluginToolkitException* - in case of any errors.

**OutputStream getOutputStream(String fileId, boolean append, int transferMode) throws PluginToolkitException;**

Gets output stream for given file for writing

Parameters:

- *fileId* - file unique identifier
- *append* - determines if content of file is overwritten
- *transferMode* - determines ascii or binary transfer (defined in PluginConstants class)

Return:

- *OutputStream* - stream created for specified file

Throws:

- *PluginToolkitException* - in case of any errors.

**void writeFile(String fileId, boolean append, int transferMode, byte buffer[]) throws PluginToolkitException;**

Writes file from byte array. This method (due to memory limitations) shall be used for small files only. The method *getOutputStream* is recommended for writing huge files.

Parameters:

- *fileId* - file unique identifier
- *append* - determines if content of file is overwritten
- *transferMode* - determines ascii or binary transfer (defined in PluginConstants class)
- *buffer* - content of file to be written

Throws:

- *PluginToolkitException* - in case of any errors.

**void writeTempFile(String name, byte buffer[]) throws PluginToolkitException;**

Writes file from byte array in temporary directory. This method (due to memory limitations) shall be used for small files only. The method *getOutputStream* is recommended for writing huge files.

Parameters:

- *name* - file name
- *buffer* - content of file to be written

Throws:

- *PluginToolkitException* - in case of any errors.

## 5.5. SECURITY

To increase the security level a plug-in developer should sign all plug-in archives, using the certificate signed by one of the Certificate Authorities, and make all plug-in files (XML and libraries) available using the *https* protocol which uses TLS (SSL) protection.



## Jar signing

The security policy is still a matter of discussion so there are no restrictive rules defining certificates that should be used for jar signing. To sign jar using the existing certificate verified by one of CA, certificate (and private key) should be first exported into a file of PKCS12 format that can be used as Java “keystore”. See <http://java.sun.com/docs/books/tutorial/deployment/jar/signing.html> for details on using the jarsigner tool (part of Sun Java SDK).

### Exporting certificate

To export certificate OpenSSL toolkit tool can be used.

Command syntax:

```
openssl pkcs12 -export -chain \  
-inkey <private key file path> \  
-in <certificate file path> \  
-out <keystore user defined name> \  
-CApath /etc/grid-security/certificates/ \  
-name <certificate alias> \  
(see open ssl documentation for details http://www.openssl.org/docs/ )
```

### Signing jar

Command syntax:

```
jarsigner \  
-keystore <keystore user defined name> \  
-storetype PKCS12 \  
-storepass <keystore passwd> \  
<jar file name> <certificate alias>
```

See <http://java.sun.com/j2se/1.4.2/docs/tooldocs/tools.html#security> for detailed jarsigner tool description.

**Fig. 11 Signing a bundle file**

## 5.6. PLUG-IN REGISTRATION

To register a plug-in within the Migrating Desktop framework the developer has to prepare an XML file containing a plug-in description. The developer may register the plug-in on his/her own, using the Migrating Desktop plug-in manager, but in that case it will be available only to him/her. The plug-ins available to all Migrating Desktop users (or to all VO members) can be registered only by the Migrating Desktop administrators, so the plug-in description has to be sent to them for acceptance and registration.

The common data required to register a plug-in:

- *plug-in type* – mandatory parameter – determines the type of the plug-in. It shall be one of the following: *APP\_VIEWER* (for application viewer plug-in), *FILE\_VIEWER* (for file viewer plug-in), *TOOL* (for tool plug-in), *JOB\_INPUT* (for job input plug-in) and *JOB\_PROCESS* (for job process plug-in);
- *symbolic name* – a mandatory parameter – a plug-in unique identifier;
- *name* – a mandatory parameter – a plug-in name in a user-friendly readable format;
- *description* – a plug-in description in a user-friendly readable format;



- *vendor* – the name of the plug-in provider;
- *version* – the current version of plug-in implementation;
- *icon image* – a plug-in image icon in the base64 format (images can be converted using several free on-line tools available on the web – e.g. <http://www.motobit.com/util/base64-decoder-encoder.asp>);
- *modification date* – date of the last plug-in modification;
- *plug-in bundles* – a mandatory parameter – shall contain at least one URL to the plug-in main bundle (a bundle containing the “activator” that starts the bundle); if the plug-in consists of a set of bundles, they can also be listed here;

Specific plug-in data depending on the type of plug-in:

- *file extensions* – parameter mandatory for file viewer plug-in – specify the list of extensions of file formats (e.g. “txt”, “gif”, “bmp”) that can be viewed using this particular plug-in;
- *infrastructure* – parameter mandatory for “job process” and “job input” plug-ins – determines infrastructure on which a specific application or a job processing tool can be run. It shall be one of the following: *LCG* or *gLite*
- *application symbolic name* – a parameter mandatory for “job input” and “application viewer” plug-ins – stands for the application unique identifier

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Sample DesktopPlugin -->
<n1:DesktopPlugin xmlns:n1="file:///E:/GRID/plugin/plugin_general.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="file:///E:/GRID/plugin/plugin_general.xsd file:///E:/GRID/plugin/plugin_general.xsd">
<!-- plugin type -->
  <type>FILE_VIEWER</type>
  <symbolicName>MyVisualiser</symbolicName>
  <info>
    <!-- Short name of the plug-in -->
    <name> Grid Benchmarks </name>
    <!-- Plug-in description -->
    <description> Desktop plug-in used for benchmarking grid systems, etc, etc, etc, </description>
    <!-- Plug-in supplier -->
    <vendor>University of Cyprus</vendor>
    <!-- Plug-in version -->
    <version>5.0.19</version>
    <!-- Plug-in icon image-->
    <iconImageBase64>
IGODlhEAAQAPcAAAQCAy4iAm5fGbeTH+S6G+3QOOBWhufgr5KQj7u2hmBgX/juyF5IAy0EAR+1
+Qs6JlX31bvv3wTlhSyECAuzOWkIEQL+2v1c0Anx0hqOOxRMDLMjFxpB28N7Ph8qUjRgDBKN9
IFJQN2qC0g5CiEPO9vYzNjQvLggyXVikAoCIEsgDszH0o+CPePj3z8wT/n6+oV1m7SQ
+JvpBIXB8fHqGXrylVB9nY2Lmg3fDt4NHR0KVxaw8EBW9EPHgXFso02l6Gk35lD5gQNV3+9uXj
AcCGFRHZLesXjEKBn3B+3ZaXisZQTEwMb52cvz3NXP1tm7UYh3Y7mXmtyuHGhcbH9gPkSXDKGa
m06rKXjicishnFREk86a1hOMuDa4DsoJCAVLR2o3llEfuLR+hEPCxgSDh8NMaiIqoJmGjo6
scu3q7vr+/cKm7Z6Jo42AXylGApF8frGRRWBdN93DcUE5RmdWfjAWB9WsQ/7++i8j
+HU9NQ8EGAoDBBkQK/Df/EkVl72uz1YmlrWogu7v7n5sl+DX6ZaFqvzx9D8VCNC+61VFH/76
/EktB9vJ8LmlXO2+Oe7R/o5+pkpNSBKODpmpalPfzPKSAPfz232ZWH2BOedS59urK/h4Xl/Tk1LaC
WarGuruvY/oOBgqaOjVhSTujO/v3672pUMLLeozV1FRm5kXv384R4OJ0k7W3dpg1o2BhgKCaOJ
as2qYMrDo/7+/iwjD31sP455sMKmetLK3ldSYG48Msu65/f2+fPx8aamng7KWWRM
+bK/louKjkt9XE7S8ZGgUKBxgKLD3O8su3gKSc7SotJyRYMet7QoOBm1mdoV4PpqHugQC
/iskJHxrY6uYx+7j1ZhUTq6XYP3+79/J+qKPISUXPQoKcm5hOxIGHzwwLvs6LwUGBcCu221e
OiyfHm9+fYqCEXGJWGgmpTS+bW/Ofs/v////////yH5BAEAAP8ALAAAAAAQABAA
JHDJm5KBCBP+m8UAg8KHBgyYEMhdYYCEgSasDawFIWlBv6JESjmRxZlP0oJvGWAKMl8
ONE9HhZx7Uz8hgN1lQqlrxUqfbx4A4pQwJARjnyeqZSKn7clCjfsGMfEGbpjQhwZ
d1ggAbRn56AY42awYDHEzk4AeHOZMuZD390fSfKlu/Jj5z9UABDgBaAwIAA7
</iconImageBase64>
    <!-- Last modification date-->
    <modificationDate>2005-12-09</modificationDate>
  </info>
  <!-- Properties of the plug-in container window DO WE NEED THIS?-->
  <window>
    <title>GridBench Viewer</title>
    <sizeX>400</sizeX>
  </window>
</n1:DesktopPlugin>
</!-- plugin type -->
```



```
<sizeY>300</sizeY>
</window>
<pluginBundles>
  <!-- Main bundle with bundle activator-->
  <mainBundle>http://ras.man.poznan.pl/~george/gridbench.jar</mainBundle>
  <!-- Mandatory parameter-->
  <!-- List of bundles required by main bundle-->
  <bundle>http://ras.man.poznan.pl/~george/jcommon-0.9.1.jar</bundle>
  <!-- Optional parameter-->
  <bundle>http://ras.man.poznan.pl/~george/xindice.jar</bundle>
  <!-- Optional parameter-->
  <bundle>http://ras.man.poznan.pl/~george/jfreechart-0.9.16.jar</bundle>
  <!-- Optional parameter-->
</pluginBundles>
<pluginProperties>
  <property name="Prop1" value="va1"/>
  <property name="Prop2" value="val2"/>
</pluginProperties>
<!-- File extensions handled by plug-in (e.g. if the plug-in is image viewer extension can be "bmp" , "jpg" , "gif" -->
<fileExtensions>
  <ext>bmp</ext>
  <ext>jpg</ext>
  <ext>svg</ext>
</fileExtensions>
<!--
<infrastructure>gLite</infrastructure>
<appSymbolicName> MAGIC</appSymbolicName>
--></n1:DesktopPlugin>
```

Fig. 12 XML registration file - example

## 5.7. CREATING A PLUG-IN “STEP-BY-STEP”

### 5.7.1. Do I need a plug-in?

The applications that have no specific requirements can be integrated within the Migrating Desktop without any additional effort, so at first the developer shall make a decision on whether the functionality currently provided by the Migrating Desktop fulfils his/her needs or should be extended.

### 5.7.2. What kind of plug-in shall I choose?

Depending on the functionality that the developer would like to add to the Migrating Desktop framework, he/she has to choose what kind of plug-in has to be implemented. The developer can choose from the types described in this document:

- *Job input plug-in* – for defining specific job input parameters;
- *Job process plug-in* – if any additional activities should be performed before submitting a job (e.g. job parameters have to be optimized before submission);
- *File viewer plug-in* – if applications produce non-standard format files that are not handled by the Migrating Desktop and that have to be presented in a user-friendly graphical format;
- *Application viewer plug-in* – for the visualisation of job results that are produced as a set of files or that have to be pre-processed (e.g. any animations, visualisation of meteorological data as weather forecast map, etc.)
- *Tool plug-in* – for adding any Java tools to the Migrating Desktop framework;

### 5.7.3. Do I need a java plug-in to specify job input parameters?

Some of the applications require specific input parameters but there are no additional relationships between those parameters. For these applications a “ready-to-use” job input plug-in (that can create



panels based on provided XSD description) may be utilized. When some additional “logic” describing the dependencies between parameters is required, the developer should implement job input plug-in on his/her own.

See appendix for the example of an XML file required for the registration of a job input plug-in interpreting XSD.

#### 5.7.4. Implementing a plug-in

Every plug-in definition consists of a set of classes/interfaces defining specific for given plug-in type: plug-in, toolkit, container and auxiliary classes (constants and factory).

- **...Plugin** – interface that contains plug-in API. It has to be implemented by the plug-in developer.
- **...PluginToolkit** – represents a plug-in toolkit providing a set of auxiliary methods;
- **...PluginContainer** – represents a plug-in container providing a set of methods which can be used for the interaction between the plug-in and the container;
- **...PluginFactory** – a factory that serves an implementation of *...PluginInterface*. It has to be implemented by a plug-in developer.
- **...PluginActivator** – a bundle activator that starts an OSGi bundle and registers a plug-in as a service;
- **...PluginConstants** – an auxiliary class that contains definitions of several constants used by a plug-in;

The developer has to create only three components from the list above: plug-in interface, factory and activator. The most important and the most sophisticated is, of course, the *...Plugin* interface implementation that provides functionality offered by a plug-in. While writing a plug-in developer may use the toolkit functionality (e.g. to gain access to resources – files, certificates, etc.). Importing is also registering plug-in as a service within a plug-in activation start() method – among the properties describing service, the plug-in symbolic name has to be specified (it is done automatically when the user extends the class *...PluginActivator*)

#### 5.7.5. Creating an OSGi bundle

Building an OSGi bundle is the next step of plug-in creation. The plug-in classes after compilation have to be packed within the Java archive file (JAR). The *jar* tool provided with Sun Java SDK can be used for that purpose. See the following link for a detailed description of *jar* tool (<http://java.sun.com/docs/books/tutorial/deployment/jar/build.html>). The created jar file has to have an OSGi *manifest* containing at least “Bundle-SymbolicName” and “Bundle-Activator” headers (see the example below). If the plug-in uses some external libraries, they must also be packed within the jar file (placing them in a separate directory will be a good idea) – in that case paths for all libraries have to be listed in header “Bundle-ClassPath”.

<b>Manifest-Version:</b>	1.0
<b>Bundle-Description:</b>	Implementation of file viewer plug-ins
<b>Bundle-Vendor:</b>	PSNC
<b>Bundle-Version:</b>	1.0.0
<b>Bundle-Activator:</b>	pl.psnk.desktop.plugins.fileviewer.impl.ViewerBundleActivator



<b>Bundle-Name:</b>	pl.psnk.desktop.plugins.fileviewer.impl
<b>Import-Package:</b>	org.apache.log4j, org.osgi.framework; version=1.2, pl.psnk.desktop.plugins.tool, pl.psnk.desktop.plugins.fileviewer
<b>Bundle-SymbolicName:</b>	pl.psnk.desktop.plugins.fileviewer.impl
<b>Bundle-ClassPath:</b>	., lib/asm/AsmVis_MD.jar, lib/pdb/jai_codec.jar, lib/pdb/pdbviewer.jar, lib/svg/apache_batik.jar

**Fig. 13 Bundle manifest - example**

### 5.7.6. *Signing a bundle*

The Java archive file containing a bundle should be signed using e.g. the *jarsigner* tool as described in chapter 5.5.

### 5.7.7. *Publishing a plug-in*

Newly created plug-in bundle(s) must be put on any network location available through the HTTP(S) protocol. Now, the XML plug-in description necessary for plug-in registration has to be prepared (see chapter 0 for details). If the developer wants to make his/her plug-in publicly available, its XML description has to be sent to the Migrating Desktop administrators for validation and registration. If it should be kept private, the developer can register it on his/her own using the Migrating Desktop plug-in manager.

## 5.8. GUIDELINES FOR PLUG-IN DEVELOPERS

### 5.8.1. *Java plug-ins*

- *Don't open a new window as the main plug-in window*

Please make the plug-in content pane the main pane of your tool. The plug-in content pane will be put in a dialog (or other Java container) created specially for that purpose. Please avoid a situation where a dialog (plug-in container) will occur empty and your tool starts its own window/frame as the main frame (however, the plug-in may open its "child" windows/frames)

- *Use `InputStreams` for reading.*

To avoid allocating a large amount of memory please use `InputStreams` for reading large remote files rather than buffers (read file sequentially, do not use the `readRemoteFile` method);

- *Use „remote streams“ carefully!*

Please be careful while using the `InputStream.read(byte[])` method: for input streams opened for "remote files" it may return data partially (as data frames come from the network), use the `InputStream.read()`

- *Store Plug-in \*.jar files in „safe“ but available network location!*



The Java archive files network location should be available via the HTTP protocol. URLs to jar's should not change.

- Write „no blocking” code – make use of threads;

Please remember that every action “fired” by any GUI event (like pressing the toolbar button, etc) is handled by Java EventQueue. Performing time-consuming operations inside EventQueue will hang the graphic (see <http://java.sun.com/docs/books/tutorial/uiswing/overview/event.html#thread>)

- Plug-in start(), stop() methods can be called in loop!
- Plug-in libraries are loaded dynamically from network location;

So all changes in the plug-in code will be available for the plug-in developer (and other users) immediately after creating the archive and putting it on the web.

### **5.8.2. Using XML schema description**

- Each XML schema file which is sent to the Migrating Desktop XML parser should be well-formed and valid according to the W3C XML schema specification. Validation and syntax can be checked using one of the tools recommended by W3C on its web pages.
- If the given element contains no “xs:appinfo” tag its label (placed on the left side in GUI) is set from the element's name.
- If the given element contains no additional “width” attribute, its width in the GUI panel is set and predefined by the Migrating Desktop XML parser .
- Check if there are any empty lines at the beginning of the XML schema file. Remove them because they can be the reason of XML parser exception – such files could not be properly parsed.
- Pay attention to the length of a single comment's line for elements in the XML schema file. They should not be too long because it does not look nice in the GUI panel with controls. If a long text is required , please split it into several lines using several “documentation” tags.
- Pay attention to the order of parameters in the application commandline (if it is important); it will be created in the same order as the order of elements in the XML schema file.
- Please put your XML schema files on the Web and make it public. Then email the Migrating Desktop Developers Team. We publish some test applet with the application parameters panel created for your XML schema file that is taken from your web page (it will be loaded via HTTP). In this way you will be able to change your XML file (by replacing it on your web page) and you will immediately see the results of your changes. So please send us the address of your web pages where we may find the newest versions of the XML schema files.



## 6. APPENDIX

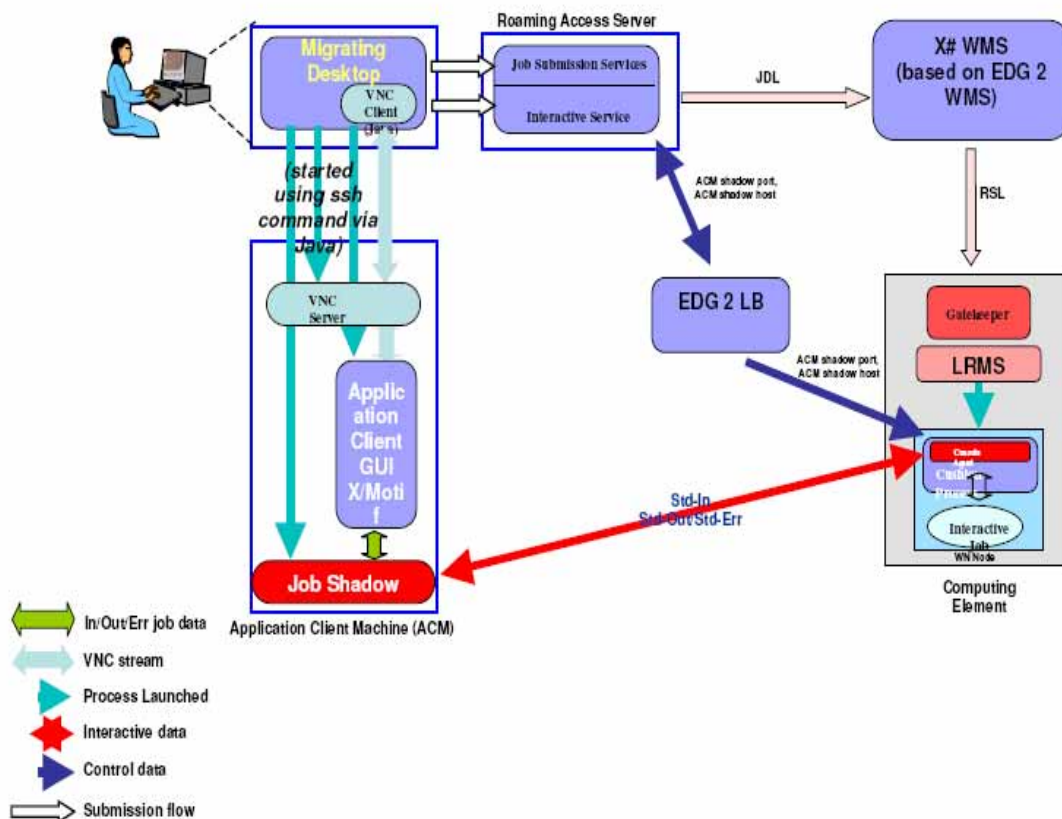
### 6.1. INTERACTIVE APPLICATION USE CASE - TECHNICAL DETAILS

Use case – submitting interactive application – the user’s point of view

#### *A. Legacy Application Case*

A sample use case of such scenario is presented in Fig. 6 and includes the following actions:

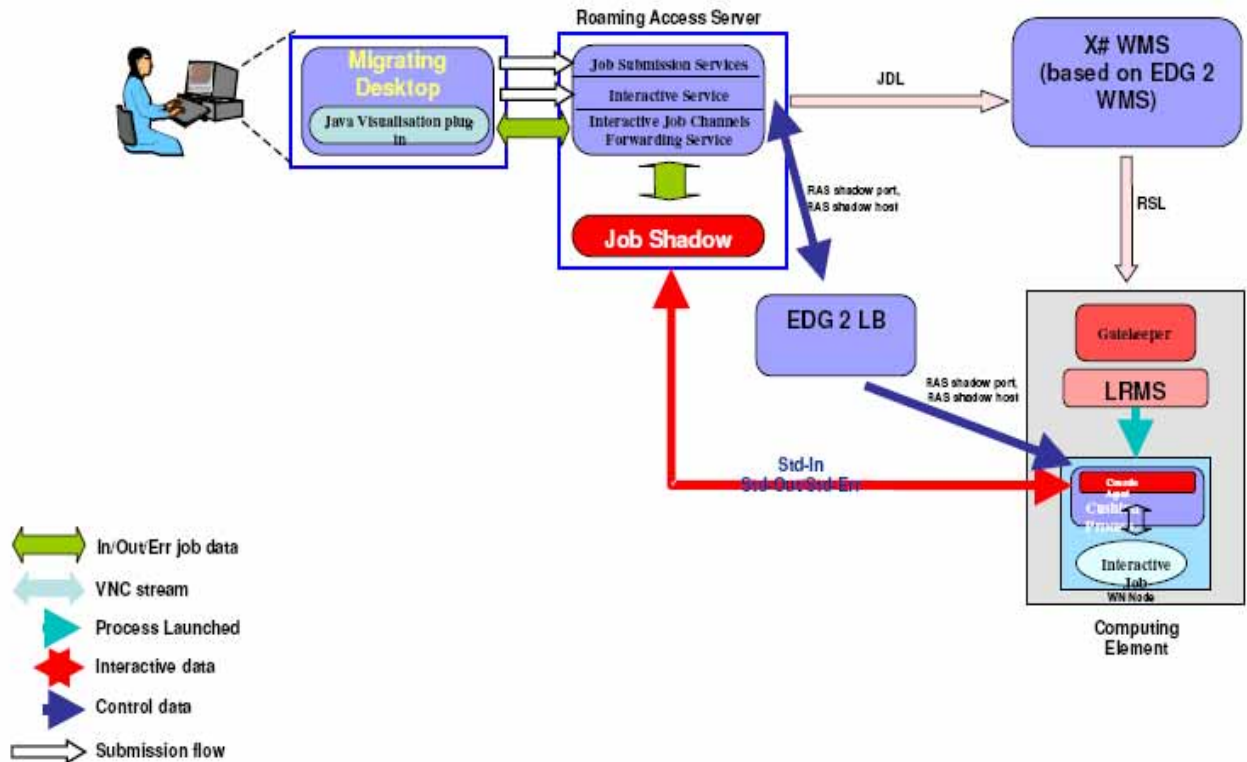
- 1) The user will prepare the interactive job submission using the Application Wizard in MD. It concerns choosing appropriate application-specific parameter input and output files and needed resources.
- 2) After submitting the job by the user, a piece of Java code inside MD will launch a script on the Application Client Machine, which will start the Job Shadow, a legacy client application and the VNC server, as a result of sending back the id's of the launched processes, the port where the Job Shadow is running and the unique name of the pipes.
- 3) The job is submitted to the Roaming Access Server. The Appropriate JDL \_le is being created on the basis of input parameters and restrictions.
- 4) The user submission request is sent to the EDG 2 WM Server component.
- 5) The EDG 2 WM Server finds an available CE that matches the requirements (including the capability to run interactive jobs) and submits the job there to be run on one of CE's WN.
- 6) The Cushion Process on WN and the Job Shadow start to exchange i/o messages until the job stops running. 7) The user using VNC from MD can see the legacy client application GUI and interact with the running job.
- 8) In case of any problem MD takes care of killing the launched process



### Java Visualization Plug-in Case

A sample use case of such scenario is presented in Fig. 7 and includes the following actions:

- 1) The user will prepare the interactive job submission using the Application Wizard in MD. It concerns choosing the appropriate application-specific parameter input and output files and needed resources.
- 2) The job is submitted to the RAS server (using the above data). The appropriate JDL file is being created on the basis of input parameters and restrictions.
- 3) RAS receives the request and starts the Job Shadow locally, adding the info on the Job Shadow port and name of the pipes to JDL
- 4) The user submission request is sent to the EDG 2 WM Server component.
- 5) The EDG 2 WM Server finds an available CE that matches the requirements (including the capability to run interactive jobs) and submits the job there to be run on one of CE's WN.
- 6) The Cushion Process on WN and the Job Shadow start to exchange i/o messages until the job stops running.
- 7) MD starts the application-related Java Visualisation client which is in touch with the Interactive Job Channels Forwarding Service on the RAS machine, which will be in charge of reading/writing the content of the relevant Job Shadow pipes.
- 8) In case of any problem MD takes care of killing the launched process.



The interactive sessions are handled by the Grid Console (GC) that is a system provided by the Condor Bypass. The Condor Bypass is used to get mostly-continuous input/output from remote programs running on an unreliable network. GC is a split execution system composed by two software components: an agent (Console Agent - CA ) and a shadow (Console Shadow - CS or Job Shadow - JS). The Console Agent runs on a Worker Node and it is a shared library that intercepts reading and writing operations on stdin, stdout, and stderr of the running job. When possible, CA sends the output back to CS. The shadow manages the input and output files according to the request of the agent. If the output sending fails, CA will instead write it on the local disk. It does not matter why the input/output operation fails; CA will keep the process running anyway. At regular intervals, it will attempt the network connection. If the connection succeeds, it will transfer any buffered data to the shadow, and then resume a normal operation.

## 6.2. PLUGIN REGISTRATION XML SCHEMA

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns="file:///E:/GRID/plugin/plugin_general.xsd" xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="file:///E:/GRID/plugin/plugin_general.xsd" elementFormDefault="unqualified"
attributeFormDefault="unqualified">
<xs:element name="DesktopPlugin">
<xs:complexType>
<xs:sequence>
```



```
<xs:element name="type">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="APP_VIEWER"/>
      <xs:enumeration value="FILE_VIEWER"/>
      <xs:enumeration value="TOOL"/>
      <xs:enumeration value="JOB_INPUT"/>
      <xs:enumeration value="JOB_PROCESS"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
<!-- Unique ID of the plug-in -->
<xs:element name="symbolicName" type="xs:normalizedString"/>
<!-- Plug-in general information-->
<xs:element name="info">
  <xs:annotation>
    <xs:documentation>Migrating Desktop Tool Plugin definition</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <!-- Short name of the plug-in -->
      <xs:element name="name" type="xs:normalizedString"/>
      <!-- Plug-in description -->
      <xs:element name="description" type="xs:normalizedString"/>
      <!-- Plug-in supplier -->
      <xs:element name="vendor" type="xs:normalizedString"/>
      <!-- Plug-in version -->
      <xs:element name="version" type="xs:normalizedString"/>
      <!-- Plug-in icon image in base64 -->
      <xs:element name="iconImageBase64" type="xs:base64Binary" minOccurs="0"/>
      <!-- Last modification date-->
      <xs:element name="modificationDate" type="xs:date"/>
      <!-- Properties of the plug-in container window-->
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="window" minOccurs="0">
  <xs:complexType>
    <xs:sequence>
      <!-- Container window title-->
      <xs:element name="title" type="xs:normalizedString"/>
      <!-- Container window horizontal size-->
      <xs:element name="sizeX" type="xs:positiveInteger" default="400"/>
      <!-- Container window vertical size-->
      <xs:element name="sizeY" type="xs:positiveInteger" default="300"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<!-- List of plugin bundles -->
<xs:element name="pluginBundles">
  <xs:complexType>
    <xs:sequence>
      <!-- Main bundle with bundle activator-->
      <xs:element name="mainBundle" type="xs:normalizedString"/>
      <!-- List of bundles required by main bundle-->
      <xs:element name="bundle" type="xs:normalizedString" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="pluginProperties" minOccurs="0" >
  <xs:complexType>
    <xs:sequence>
      <xs:element name="property" minOccurs="0" maxOccurs="unbounded">
```



```
<xs:complexType>
  <xs:attribute name="name" type="xs:normalizedString"/>
  <xs:attribute name="value" type="xs:normalizedString"/>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="fileExtensions" minOccurs="0">
  <!-- Required for FILE_VIEWER-->
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ext" type="xs:normalizedString" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="infrastructure" default="gLite" minOccurs="0">
  <!-- Required for JOB_PROCESS JOB_INPUT-->
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="gLite"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
<xs:element name="appSymbolicName" type="xs:normalizedString" minOccurs="0"/>
  <!-- Required for JOB_INPUT APP_VIEWER-->
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

**Fig. 14 Plug-in registration XML schema**

### 6.3. EXAMPLE OF XML SCHEMA JOB INPUT PANEL SPECIFICATION

Below you will find an example XML schema code (with a set of GUI tabs and controls) and GUI panel screenshot created by the MD XML parser.

More XML schema examples you can find at: <http://ras.man.poznan.pl/crossgrid/xml/>

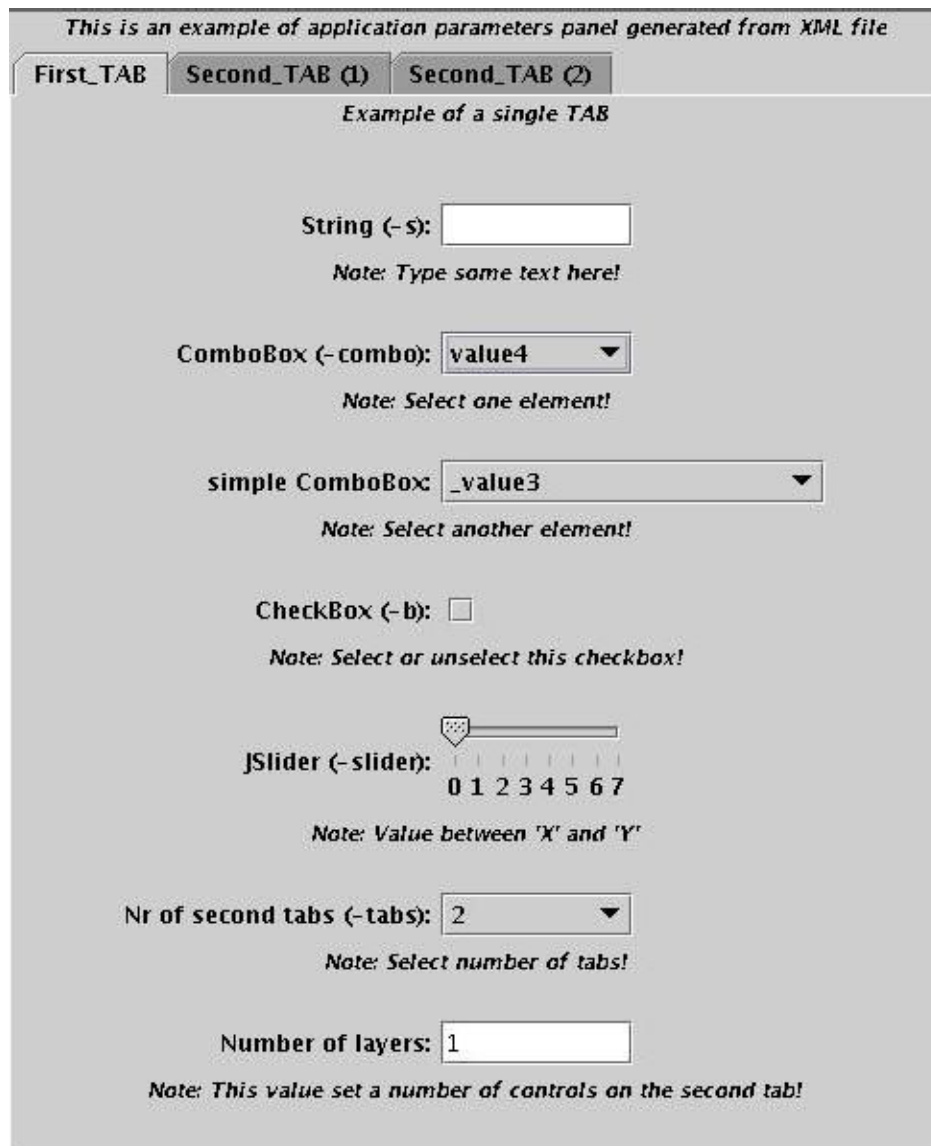


Fig. 15 GUI panel screenshot created by the MD XML parser.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <!-- TYPE DEFINITIONS -->
  <xs:simpleType name="MyComboType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="value1"/>
      <xs:enumeration value="value2"/>
      <xs:enumeration value="value3"/>
      <xs:enumeration value="value4"/>
      <xs:enumeration value="value5"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="TABS">
    <xs:restriction base="xs:integer">
      <xs:enumeration value="0"/>
      <xs:enumeration value="1"/>
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```



```
<xs:enumeration value="2"/>
<xs:enumeration value="3"/>
<xs:enumeration value="4"/>
<xs:enumeration value="5"/>
</xs:restriction>
</xs:simpleType>
<!-- ***** -->
<!-- ELEMENTS -->
<xs:element name="XML_Parser_Test">
  <xs:annotation>
    <xs:appinfo>XML Parser Test</xs:appinfo>
    <xs:documentation>This is an example of application parameters panel generated from
XML</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="First_TAB">
        <xs:annotation>
          <xs:appinfo>TAB 1</xs:appinfo>
          <xs:documentation>Example of a single TAB</xs:documentation>
        </xs:annotation>
        <xs:complexType>
          <xs:sequence>
            <xs:element name="str1">
              <xs:annotation>
                <xs:appinfo>String</xs:appinfo>
                <xs:documentation>Type some text here!</xs:documentation>
              </xs:annotation>
              <xs:complexType>
                <xs:attribute fixed="-s" name="prefix" type="xs:string" use="required"/>
                <xs:attribute fixed="100" name="width" type="xs:integer" use="optional"/>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<!-- ***** -->
<!-- ComboBox example -->
<xs:element name="Box">
  <xs:annotation>
    <xs:appinfo>ComboBox</xs:appinfo>
    <xs:documentation>Select one element!</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="MyComboType">
        <xs:attribute name="width" fixed="100"/>
        <xs:attribute name="prefix" fixed="-combo"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
<!-- ***** -->
<!-- Simple ComboBox example -->
<xs:element name="Box2">
  <xs:annotation>
    <xs:appinfo>simple ComboBox</xs:appinfo>
    <xs:documentation>Select another element!</xs:documentation>
  </xs:annotation>
  <xs:simpleType name="MyComboType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="_value1"/>
      <xs:enumeration value="_value2"/>
      <xs:enumeration value="_value3"/>
      <xs:enumeration value="_value4"/>
      <xs:enumeration value="_value5"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```



```
</xs:restriction>
</xs:simpleType>
</xs:element>
<!-- ***** -->
<!-- CheckBox (boolean value) example -->
<xs:element name="Bool">
  <xs:annotation>
    <xs:appinfo>CheckBox</xs:appinfo>
    <xs:documentation>Select or unselect this checkbox!</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:boolean">
        <xs:attribute name="width" fixed="100"/>
        <xs:attribute name="prefix" fixed="-b"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
<!-- ***** -->
<!-- JSlider (range of values) example -->
<xs:element name="range">
  <xs:annotation>
    <xs:appinfo>JSlider</xs:appinfo>
    <xs:documentation>Value between 'X' and 'Y'</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:integer">
        <xs:attribute fixed="-slider" name="prefix"/>
        <xs:attribute fixed="100" name="width"/>
        <xs:attribute name="rangeMin" fixed="0"/>
        <xs:attribute name="rangeMax" fixed="7"/>
        <xs:attribute name="rangeStep" fixed="1"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
<!-- ***** -->
<!-- Dynamic XML example -->
<xs:element name="Box2" default="2">
  <xs:annotation>
    <xs:appinfo>Nr of second tabs</xs:appinfo>
    <xs:documentation>Select number of tabs!</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="TABS">
        <xs:attribute name="width" fixed="100"/>
        <xs:attribute name="prefix" fixed="-tabs"/>
        <xs:attribute name="reference" fixed="Second_TAB"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
<xs:element name="set">
  <xs:annotation>
    <xs:appinfo>Number of layers</xs:appinfo>
    <xs:documentation>This value set a number of controls on the second tab!</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:attribute name="reference" fixed="dyn"/>
    <xs:attribute name="width" fixed="100"/>
  </xs:complexType>
</xs:element>
```



```
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
<!-- ***** -->
<!-- Second TAB of controls (empty) -->
<xs:element name="Second_TAB" minOccurs="0" maxOccurs="10">
  <xs:annotation>
    <xs:documentation>Another example of TAB</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <!-- Dynamically changed control -->
      <xs:element name="dyn" minOccurs="0" maxOccurs="10"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

Fig. 16 An example of XML schema code

#### 6.4. EXAMPLE OF JOB INPUT “READY-TO-USE” PLUGIN

```
<?xml version="1.0" encoding="UTF-8"?>
<n1:DesktopPlugin xmlns:n1="file:///E:/GRID/plugin/plugin_general.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="file:///E:/GRID/plugin/plugin_general.xsd file:///E:/GRID/plugin/plugin_general.xsd">
  <type>JOB_INPUT</type>
  <symbolicName>ANN_HEP</symbolicName>
  <info>
    <name> Neural Network HEP Application </name>
    <description>
      This application is used to train an Artificial Neural Network using simulated data for the DELPHI
      experiment. The ANN is trained to distinguish between signal (Higgs boson) events and background
      event (in the demo the background used includes WW and QCD events). The evolution of the training
      can be monitored using the MD through a graphics error, and 4 small graphics that show each of them
      the ANN value vs. an event variable (that can be selected by the user). The application is available
      compiled with MPICH-P4 for intracluster use and with MPICH-G2 for intercluster use. This application
      uses the interactive input channel to let the user make a clean stop of the training (instead of killing the
      job), and also the possibility of resetting the ANN weights to random values, to avoid local minima.
    </description>
    <vendor>Instituto de Física de Cantabria</vendor>
    <version>1.0.1</version>
    <iconImageBase64>
      R0IGODIhEAAQAPcAAAQCAy4iAm5fGbeTH+S6G+3QOOBWhufgr5KQj7u2hmBgX/juyF5IAy0EAR+1
      np+Qs6JIX3l1bvV3wTihSyECAuzOWkIEQL+2v1c0Anx0hqOOxRMDLMjFxpB28N7Ph8qUjRgDBKN9
      GldFcvX88IFJQN2qC0g5CiEPO9vYzNjQvLGgyXVikAoCIEsgDszH0o+CPePj3z8wT/n6+oV1m7SQ
      hJ+JvpBIXB8fHqXrylVB9nY2Lmg3fDt4NHR0KvXaw8EBW9EPHgXFsO02l6Gk35iD5qONv3+9uXj
      6AcCGFRH2LesXjEKB3B+3ZaXisZQTEwMb52cvz3NXP1tm7UYh3Y7mXMtyuHGhcbH9gPksXDKGa
      eKeEZ8m06rKXjicishnFREk86a1hOMuDa4DsoJCAVLr2o3lleFuLR+hEPCxgSDh8NMailqoJmGjo6
      MdnKoQYCEm5scu3q7vr+/cKm7Z6Jo42AXylGApF8fGRRWBdN93DcUE5RmdWfjAWB9WsQ/7++i8j
      Of72+HU9NQ8EGAoDBBkQK/Df/EkvL72uz1YmIrwogu7v7n5sl+DX6ZaFqvz9D8VCNC+61VFH/76
      /EktB9vJ8LmIXO2+Oe7R/o5+pkpNSBkODpmaIpFzPKSAPfz232ZWH2BOedS59urK/h4XI/Tk1LaC
      e15WarGuruvY/oOBgqaOjVhSTujO/v3672pUMLLeozV1FRm5kXv384R4OJ0k7W3dpg1o2BhgKCaOJ
      hTsnAxAKCc64as2qYMrDo/7+/iwjD31sP455sMKmetLK3ldSYG48Msu65/f2+fPx8aamnkg7KWrm
      GLOIm+bK/louKjpt9XE7S8ZGgUKBxgKld3O8su3gKsck7SotJyRYMet7QoOBm1mdoV4PpqHugQC
```



## PROCEDURES OF INTEGRATING APPLICATIONS WITH THE MIGRATING DESKTOP

```
Dvrk/iskJHxrY6uYx+7j1ZhUTq6XYP3+79/J+qKPISUXPQoKcm5hOxIGHzwwLVs6LwUGBcCu221e  
gCINBsaEf5OIYfHm9+fYqCEXGJWGgmpTS+bW/OfS/v//////////yH5BAEAAP8ALAAAAAQABAA  
AAigAP8JHDjrn5KBCBP+m8UAg8KHBgyYEMhDYyceEgSasDawFIWlIbv6JESjmRxZIP0oJvGWakMI8  
fvzMQwhHoDZ5ONE9HHhzX7Uz8hgN1IQQlrxUqfbxA4pQwUARjnyeqZSKn7cLCjfsGMfEGbpjQhwZ  
UpgEVgw6d1ggAbRn56AY42awYDHEzk4AeHOZMuZD390fSfKlu/Jj5z9UABDgBaAwIAA7
```

```
</iconImageBase64>
```

```
<modificationDate>2005-02-14</modificationDate>
```

```
</info>
```

```
<pluginBundles>
```

```
<mainBundle>http://ras.man.poznan.pl/plugins/XMLplugin.jar</mainBundle>
```

```
</pluginBundles>
```

```
<pluginProperties>
```

```
<property name="XmlDescription" value="http://grid.ifca.unican.es/user/ANN/NeuralNetwork.xsd"/>
```

```
</pluginProperties>
```

```
<infrastructure>gLite</infrastructure>
```

```
<appSymbolicName>ANN_HEP</appSymbolicName>
```

```
</n1:DesktopPlugin>
```