Grant agreement no. 283562

N4U

NeuGRID for you:

expansion of NeuGRID services and outreach to new user communities

Combination of Collaborative Project and Coordination and Support Action

Objective INFRA-2011-1.2.1 – e-Science environments

Start date: July 1st 2011**- Duration**: 42 months

Deliverable data

Deliverable reference number and title: D5.2 DCI Pipeline APIs and Testbed Provision Report
Due date: month 12, 30 June 2012
Actual submission date: 27 July 2012
Lead contractor for this deliverable: P2 maatG
Dissemination level: public

Authors

Jérôme Revillard (P2 maatG)
Baptiste Grenier (P2 maatG)
R.A. van Rchijndel (P4 VUA)

Approval

Workpackage Leader: Jérôme Revillard (P2 maatG)
Project Coordinator: G.B. Frisoni (CO1 FBF)
PMT members: G.B. Frisoni (CO1 FBF), D. Manset (P2maatG), R. McClatchey (P3 UWE)

History Record

Version	Date	Notes
0.1	02/06/2012	maatG prepared the 1st draft
0.2	05/06/2012	maatG added the architecture section
0.3	10/06/2012	maatG add references + consolidate section 2.3
0.4	20/06/2012	maatG misc updates
0.5	25/06/2012	maatG integrated the SRP information
1.0		Integration of the VUA expressLane part

Table of Contents

1. Introduction

1.1. Purpose of the document

This document reports on the DCI Pipeline Service APIs which are the ExpressLane2DCI API and the VirtualLab2DCI API. This report will be destined to developers from WP6/7/8/9 and 10, as a reference document for using the infrastructure whenever managing, discovering, publishing and running algorithms and pipelines/workflows.

Prior to reading this document the reader should be familiar with the following deliverables produced within the neuGRID project, which have or are considered to potentially impact on the design and developments of the DCI Pipeline APIs:

- neuGRID WP6 - Distributed Medical Services Provision: D6.1. Design Document including API Documentation and Description of Functionality for the Underlying Layer
- neuGRID WP7 - Grid Services Provision: D7.1. Test-bed Installation and API Documentation

1.2. Document organisation

This document will be mainly composed of two different parts: the first part will present the DCI Pipeline APIs (ExpressLane2DCI and VirtualLab2DCI) in detail and the second part will provide some information about the test-bed that is in place.

1.3. Document review

According to the Description of Work (DoW), an update of this document will be released in June 2013 (M24) and December 2015 (M42) respectively under the names of *D5.4 DCI Pipeline APIs and Testbed Provision Report* and *D5.6 DCI Pipeline APIs and Testbed Provision Report*. These documents will be mainly focused on updating the developers about APIs modifications.

2. The DCI Pipeline APIs

2.1.ExpressLane2DCI API

Introduction

In medical research most computational demanding work is trivial parallel, for example calculating the brain volume of 400 patients and 400 controls. Whatever algorithm is used for calculating the brain volume, all 800 brains can be calculated in parallel.

ExpressLane is designed to make is practical to run a given script on multiple scans or datasets. ExpressLane is designed to submit jobs to the DCI just by generating a list of datasets without having knowledge of the underlying job submission system. The datasets used by ExpressLane can be in various directories. Within ExpressLane a number of search directories can be specified in which the datasets are searched for. The status of a job and its output are automatically sent to a given output directory. All directories used for input and output are specified in the ".express" file (e.g. Sienax.express for an Expresslane setup that runs Sienax)

Files to interface with ExpressLane

To interface with ExpressLane four configuration files are used. Most users will only alter the ".express" file and the ".jobs" file. The types of files to interface with ExpressLane are described in the following paragraphs.

2.1.2.1. Files to interface with ExpressLane

The ExpressLane ".express" file contains the name of the script to be run. This name is expected in the ".express" file after "script=". The name of the actual script file to be run will be the given name followed by the extension ".script". In the ".express" file below this will be Sienax.script.

```
# Demo of ExpressLane express file for Sienax
script=Sienax
jobs=Sienax.jobs
inputDir=lfn:/grid/vo.neugrid.eu/data/US-ADNI-DICOMS-ZIP
inputDir=lfn:/grid/vo.neugrid.eu/data/US-ADNI-DICOMS-ZIP2
outputDir=lfn:/grid/vo.neugrid.eu/home/O=dutchgrid_O=users_O=vu_OU=vumc_CN=Ronaldus_Antonius_v
an_Schijndel/dir/SienaxUserOutput
commonDir=lfn:/grid/vo.neugrid.eu/home/O=dutchgrid_O=users_O=vu_OU=vumc_CN=Ronaldus_Antonius_v
an_Schijndel/sienax_express/CommonDir
ExplicitFiles=1
MemRequired=1GB
DiskSpaceRequired=4GB
```

The ExpressLane ".express" file is also expecting to have a line starting with "jobs=" followed by the name of the jobs file. The format of these jobs file is specified section 2.1.2.2 below.

The idea of ExpressLane is to have a list of input directories in which the datasets or files are searched for (like the %PATH% in DOS or $path in linux). These input directories are given in the ".express" file in lines holding "inputDir=". These directories will be used as a search path for files specified in the jobs file. If the datasets or files have to be searched in more than one directory, the ".express" file does contain more than one "inputDir=" lines.

Besides the input directories also an output directory has to be specified. This is done on a line starting with "outputDir=" followed by the output directory.

If there are files that are needed by all jobs they can be put in a directory specified after the “commonDir=” line. More than one commonDir can be specified and all files in the given directories will be copied to the sandbox of the computing element before the job is started (for the jobs itself it will look whether these files are in the directory in which the job script is started).

The explicit files line starting with “ExplicitFiles=” followed by the number of explicit files, specifies the number of arguments that should be searched for in the input directories. The parameters after the number of explicit files will not be copied from the input directories. These parameters can for instance be used to specify options for the program.

The MemRequired and DiskSpaceRequired are directly copied to the underlying job submission system.

2.1.2.2. The “.jobs” file

In the “.express” file a jobs file have to be specified. An example of a jobs file is given below.

```
# Example jobs file for Sienax
  037_S_1421SNA        AdniNG+037_S_1421+20070827+094649+S38602+MPR0+MCI.tar.gz
  127_S_1427SNA        AdniNG+127_S_1427+20070820+130816+S37933+MPR0+MCI.tar.gz
  027_S_1213SNA        AdniNG+027_S_1213+20070119+144325+S25493+MPR0+MCI.tar.gz
  029_S_1218SNA        AdniNG+029_S_1218+20070123+123019+S25478+MPR0+MCI.tar.gz
```

The first name in each line of a jobs file is the jobs name. All files staring with this jobs name found in any of the input directories will be copied to the sandbox directory of the computing element for the given job.

Also all files starting with one of the explicit filenames are copied from the input directory to the sandbox directory of the computing element for the given job.

2.1.2.3. The “.script” file

The “.script” file (e.g. Sienax.script for Sienax) will contain the actual script to be run. Most users will use “.script” files that have already been generated. Users who want to program their own script in a script language can generate their own scripts.

2.1.2.4. The “.ext” file

The “.ext” file (e.g. Sienax.ext for Sienax) contains information directly passed to the job scheduler to be able to specify which hardware to be used, which packages should be available etc. Users which aren’t programming themselves will use the “.ext” file that is supplied.

Output files

The output directory is specified in the “.express” file. In this output directory a number of files are written. These files can be devided into three categories. The status files, the standard out and standard error file and the files generated by the job.

2.1.3.1. Status files

The script that handles the ExpressLane interface writes 3 status files to the output directory given in the “.express” file, to monitor its status:

- “<jobname>.1wrapper-started”: which is generated when the wrapper script on the CE is started.
- “<jobname>.2script-launched”: which is generated when the actual script on the CE is launched.
- “<jobname>.3script-finished”: which is generated when the actual script on the CE is finished.

The script itself is expected to generate a “<jobname>.4done” file when it is successfully finished, which is described below in the paragraph “Job output”.

2.1.3.2. stdout and stderr

The stdout and stderr of the job to be run will automatically be copied to the given output directory under the name “<jobname>.stdout” and “<jobname>.stderr”. This will make it easy to check for errors and correct or incorrect workflow for a given job.

2.1.3.3. Job output

All files in a job that are written to the directory Ouput (which have to be generated by the script) will be automatically copied to the given output directory. To keep the number of files limited it might be handy to zip a number of generated files for quality check and put this zip file in the output directory.

As already mentioned under the paragraph “Status files” the script is expected to generate a file “<jobname>.4done. This file should also be generated in the output directory.

Running ExpressLane

The ExpressLane is launched by calling “interchange.sh” with the name of the express file. E.g.

```
interchange.sh Sienax.express
```

“interchange.sh” is a script that was created in N4U in order to translate the EpressLane formatted jobs into JDL jobs. It will generate a JDL job file for each line in the “.jobs” file. This JDL jobs file will be sent to the scheduler (WMS) to be run on one of the computing elements.

The script “interchange.sh” will generate a “.jids” file (e.g. Sienax.jids). These files contain the WMS job ids for the submitted files.

Re-running ExpressLane

Expresslane contains a special feature for rerunning jobs that have failed for whichever reason. If the script that is launching the ExpressLane is relaunched it will run all jobs without an “<jobname>.4done” file. The “<jobname>.4done” file will have to be generated by the script when the jobs were executed successfully (see also the paragraphs about “Status files” and “Job output”).

Testing ExpressLane

To test ExpressLane the “Sienax” script can be used on the ADNI dataset to run FSL Sienax on this data. The advantage of Sienax is that it doesn’t run long (approximately 10 minutes)

but does some real calculations. This makes it very useful to test job submission and job resubmission on a large dataset without having to wait for days before the test is completed.

2.2.VirtualLab2DCI API

Introduction

The role of the VirtualLab2DCI APIs is to provide application developers a way to create and design workflow, in a workflow authoring environment of their choice. The current implementation consists of the following implemented features: 1) A fully defined web service interface of the Pipeline Service; 2) An implemented translation component that supports translations from the previous API to the SAGA [1] one and WS-BPEL [2]; 3) An enactor that uses the JSAGA [3] implementation of the SAGA API for submission to the grid.

Architecture

2.2.2.1. Background

The N4U architecture is derived from the neuGRID one. As you can see in the D10.1 of the neuGRID project [4], the Service Oriented Architecture (SOA) paradigm was followed. The main characteristics of a SOA are the loose coupling between services, the abstraction from technological aspects and its extensibility; features considered essential to cope with distributed developments, heterogeneous technologies integration and to leverage multi-partners collaborations.

As you can see in the Figure 1, starting from the very bottom of the system, i.e. “Backends Middleware”, with IT legacy assets to be used in the project such as grid and database infrastructures, various abstraction levels are then introduced. The most important one, so-called “Backends Abstraction”, aims to wrap up underlying backends and to allow partners to develop grid/database agnostic software while still interacting with given technologies and corresponding specificities. Based on this abstraction, further layers are superimposed which deliver more and more specific functions as distance to end-users shrinks. As such, “Domain Logic” aims at grouping so-called “medical generic services” – e.g. medical querying, medical data acquisition and quality control etc – which could be reused in other medical fields, whereas “Business Logic” only focuses on Neuro-Sciences features, e.g. cortical thickness pipeline, segmentation/ normalization algorithms etc, which are then accessed by end-users through a dedicated web portal exposing specialized interfaces.

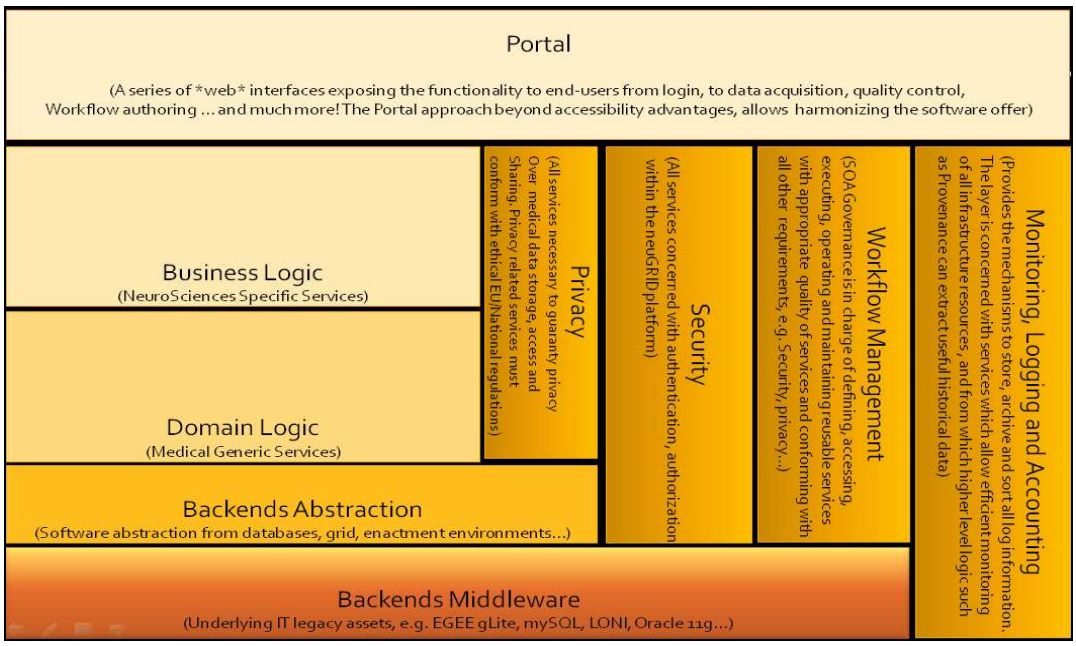

Figure 1: neuGRID system architecture - Layer level

Orthogonally, the platform aims to offer various means to trace system activity via the "Monitoring, Logging and Accounting" subset of services. The neuGRID services were delivered within a secure environment implementing a security scheme as dictated by the requirements, hence having a "Security" layer spanning from the top business logic, to domain logic and finally underlying abstraction. The same applies for privacy aspects since dealing with sensitive data and applications, though essentially impacting on both business and domain logics.

As stated in this D10.1 deliverable, "*From WP10 standpoint, key elements of this architecture lie in the "Workflow Management" layer, where the necessary logic for publishing, discovering and composing functionality is expected to materialize, most likely under the form of service utilities.*". The result at the end of neuGRID was the creation/integration of 3 services:

- The Pipeline Service [5] which was meant to expose a simple API to run/monitor workflows of different types: Grid or pure Web Services workflows.
- The Grimoires Service [6] which provided the publication/discovery API.
- The Orchestration Service which was mainly an ActiveBPEL wrapper which was used by the Pipeline service in order to orchestrate Web Services workflows.

2.2.2.2. The Pipeline service architecture

2.2.2.2.1. History

The N4U project faced a significant problem at the very beginning with Globus Toolkit. Indeed the neuGRID gateway was mainly built on top the Globus Toolkit v4.2.1 (java version) and a news was published at the end of the year 2009 saying that the java version will be stopped (http://www.globus.org/news.html#161) and that a new project named Crux [7] will follow but it does not occurs.

In order to build the N4U system on a reliable technology, the decision to migrate the all system to the Apache CXF framework [8] was taken. Apache CXF is an open source services framework. It helps you build and develop services using frontend programming APIs, like JAX-WS and JAX-RS. These services can speak a variety of protocols such as SOAP, XML/HTTP, RESTful HTTP, or CORBA and work over a variety of transports such as HTTP, JMS or JBI. This is a big advantage compare to the Globus which was only able to expose SOAP web services over HTTP. Also, this project is very active and has a large community (integrated with many other Apache projects) which is something important not to face the same problems as for Globus.

The core of the system had completely been rewritten as well as all the services. This migration took more than six month to be done and a full year was needed to have something stable enough for a production environment.

Last but not least, the Grimoires and Orchestration services did not pass the migration because both ActiveBPEL and Grimoires were no more maintained. At the time of writing, now that the core migration is done, investigations are done to find good replacements.

2.2.2.2.2. Current architecture

The main service that is used is the Pipeline Service. According to the neuGRID design philosophy the Pipeline Service has been designed around a Service Oriented Architecture (SOA). The web service binding enables clients to interact with the Pipeline Service and perform various functions such as submission of workflows, tracking progress and various control functions. There are essentially two types of methods supported in the Pipeline Service. The submission method exposes functionality that enables a user to submit a workflow. Workflow Control methods expose functionality that enable interaction with currently executing workflows. The current architecture of the service is the following:

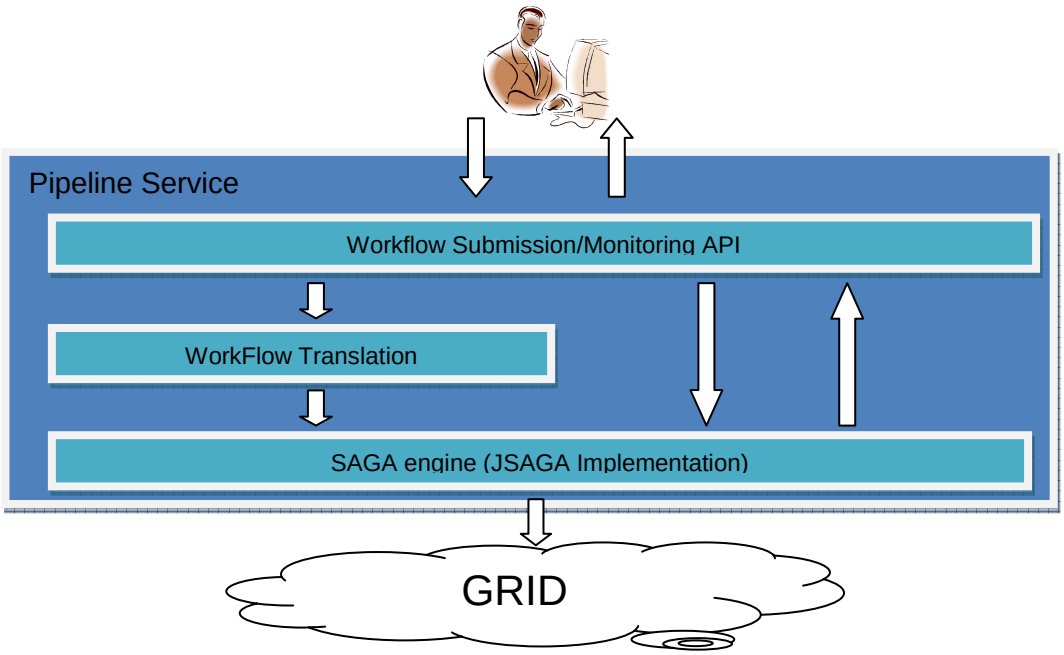

Figure 2: Actual system architecture

The Pipeline service exposes a "*Workflow Submission/Monitoring API*" that is accessible using regular web service technology (Apache CXF Framework). This API will be explained in detail in section 2.2.3. It will explain also how a workflow is represented.

When a workflow is submitted, the first action that is done is to translate it into the SAGA representation. This is the role of the "*Wokflow translation*" step.

Once done, the SAGA engine (JSAGA in our case) is able to submit the workflow into the GRID. This same API is then used in order to control the job execution.

With this design, into neuGRID, the users were able to submit real workflows to the Grid middleware but unfortunately, this is not the case anymore. Indeed, the support for Directed Acyclic Graph (DAG) jobs (Handled previously in the gLite WMS by the Condor DAGMan [9]) has been dropped in the new Computing Elements named CREAM [10]. In the next section, you will see how the architecture will be updated in order to cope with this.

2.2.2.2.3. The evolution

As you can see, in the previous section, the design that was done in the neuGRID project has some problems due to many historical changes in the community. The WP5 team is currently looking at a new architecture which will solve those issues. What is planned to do actually is to integrate a WS-BPEL Workflow Engine inside the Pipeline service.

The new architecture will be the following (c.f. Figure 3):

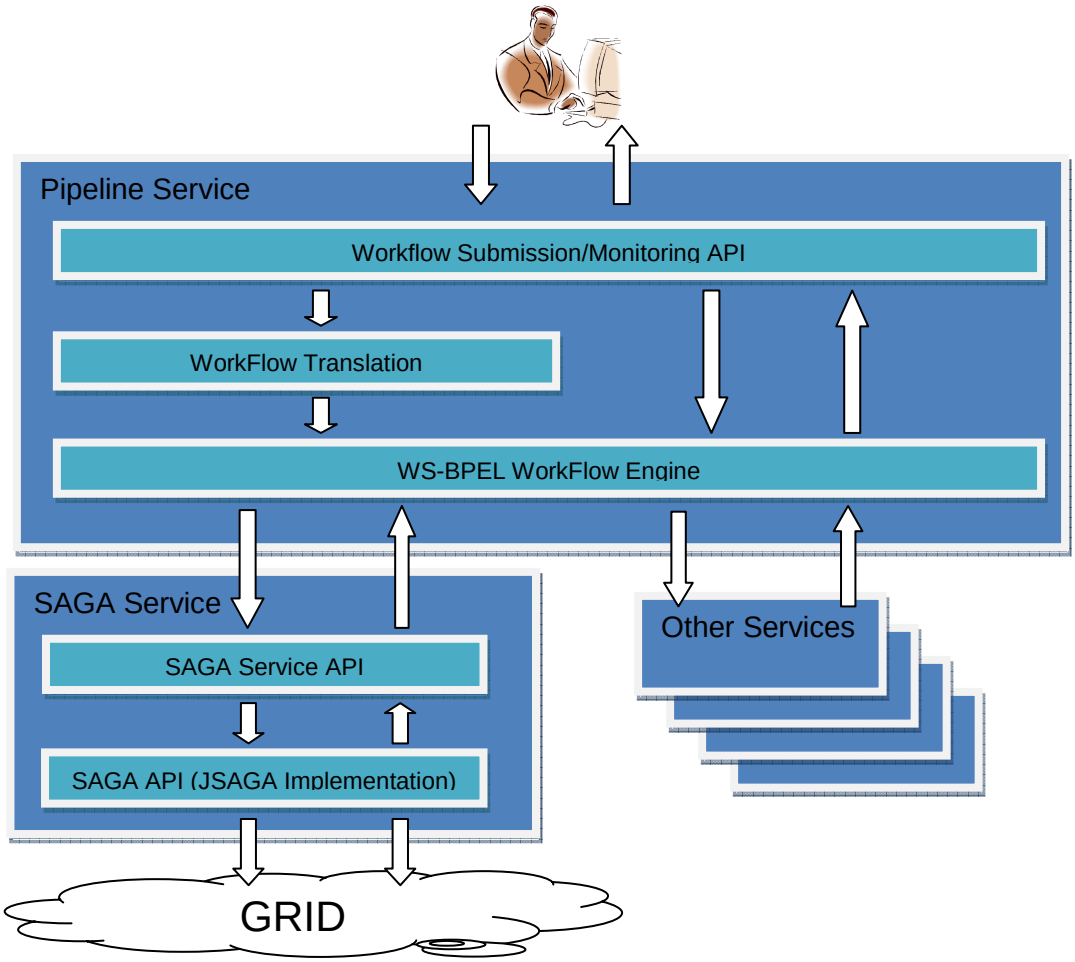

Figure 3: The new architecture

As you can see, instead of translating the workflow into a SAGA representation a WS-BPEL representation will be generated. This later will be submitted to an embedded *workflow engine*.

If the workflow is composed of GRID jobs then the SAGA service will be called for each of them. This service exposes an API which is close to the real SAGA API and which is already available in the system and used for instance by the Grid Browser application [11].

There are mainly two advantages to this architecture compared to the previous one:

- Real GRID jobs workflows can be submitted again.
- As the WS-BPEL engine is able to deal with all the web services, composition of different kind of jobs can be done: much more than just GRID jobs.

Implementation

2.2.3.1. Exposed methods

As described in the section 2.2.2.2.2, the Pipeline API exposes currently the methods to submit and monitor a Workflow. Bellow, the different methods are described and you can find the real WSDL of the Pipeline service in Annex 1:

- **public String workflowSubmit(Workflow workflow):**

As indicated by his mane, this method aims at submit workflows. The workflow object oriented description is presented in section 2.2.3.2.

- **public WorkflowStatuses listWorkflows(String regex):**

This method provides a way to list the workflows that are currently monitored by the Pipeline service. The *regex* parameter can be used to filter the result based on the workflow ID. The regex format is the one used in the Java Pattern class: http://docs.oracle.com/javase/1.4.2/docs/api/java/util/regex/Pattern.html

- **public boolean cancelWorkflow(CancelWorkflow cancelWorkflow):**

This method allows cancelling a monitored workflow. The *CancelWorkflow* object has two attributes: the first one is the *workFlowID* and the second one is a Boolean named *force*. If force is *true* then the workflow is purged from the pipeline service even if the underlying tasks were not properly cancelled.

- **public WorkflowStatus getWorkflowStatus(String workflowID):**

This method simply allows getting the status of a job which have the *workflowID* ID. As you can see in the WSDL, the different statuses are:

- *SUBMITTED: the job was just submitted to the underlying system.*
- *QUEUED: the job is queued in the underlying system.*
- *RUNNING: the job is running in the underlying system.*
- *DONE: the job is finished (for the Grid middleware, this does not mean that the job does not finish with an exit code different from 0.).*
- *CLEARED: the job output was already retrieved and is no more available.*
- *CANCELED: the job was cancelled.*

- *FAILED_ERROR: The job failed because there was a problem in the underlying infrastructure.*
- *FAILED_ABORTED: The job was aborted because of a problem in the underlying infrastructure*
- *SUSPENDED_QUEUED: Not used*
- *SUSPENDED_ACTIVE: Not used*
- *UNKNOWN: The job status cannot be determined.*

- **public WorkflowOutput getWorkflowOutput(String workflowID):**

If the job is in the DONE state, this method allows retrieving the output of the job which has the *workflowID* ID.

- **public boolean purgeWorkflow(String workflowID)**

This method can be used to clean up the Pipeline service from the different temporary files of a specific workflow.

2.2.3.2. The objected oriented workflow representation

This representation was created having in mind to keep it simple but powerful enough for the need of the project. Based on the neuGRID experience, the following class diagram was designed:

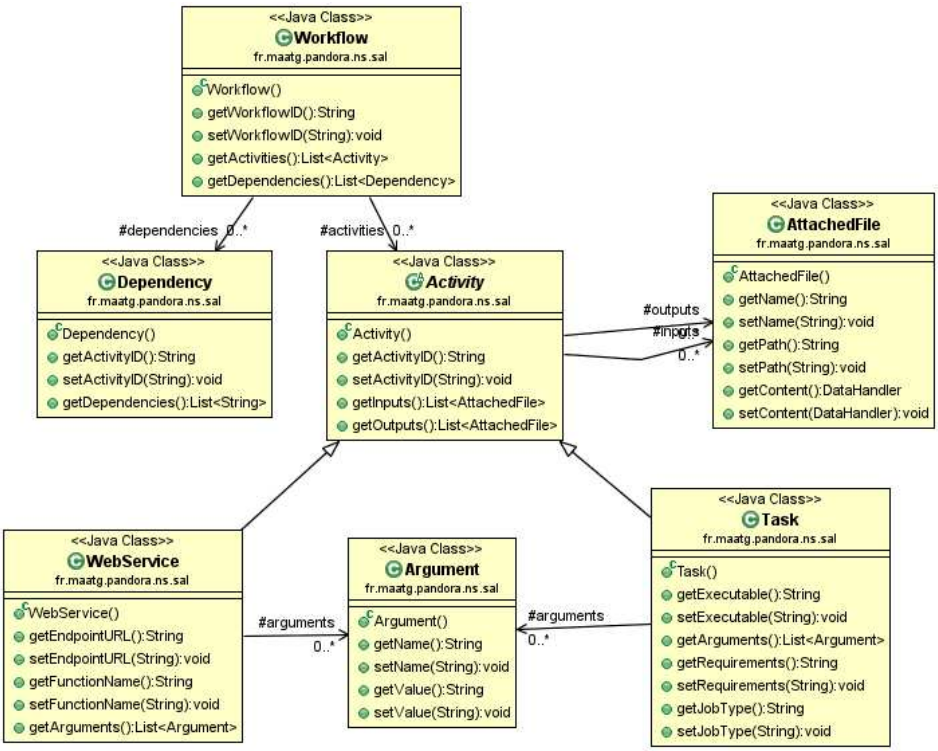

Figure 4: Class diagram of the object oriented workflow representation

2.2.3.2.1. Worflow

The *Workflow* object is characterised as a set of *Activities* which are linked together by some *Dependencies.*

2.2.3.2.2. Activity

This class represents all the entities that are manipulated inside a *Workflow*. It contains common attributes like the *ActivityID*.

2.2.3.2.3. Dependency

The *Dependency* class defines a structure for declaring a dependency. The basic properties in this class include an *ActivityID* property which defines the ID of the task concerned. The other property is the dependencies property which is a list of *ActivityID*, which enumerates all IDs of activities which have a dependency relationship with the activity.

2.2.3.2.4. Task

The *Task* class contains properties that define an executable task in a Workflow. It is a sub-class of the *Activity* one and it contains the properties needed to create a concrete Grid job using the SAGA API internally.

2.2.3.2.5. Web service

This class is, as the *Task* class, a sub-class of the *Activity* one. This is the concrete representation of a web service activity.

Conclusion

Despite the problems that occur even before the beginning of the N4U project that forced the WP5 team to migrate to the Apache CXF Service Framework, the Pipeline API is now stable in production. It has already been integrated with the LONI Pipeline Server [14] using there plugins mechanism.

The entire stack has been validated by the “LInked Neuroscientific Grand challenge” (LINGA) data challenge [15]. LINGA aims at elaborating on an existing set of neuroscientific application toolkits (primarily focused on Alzheimer’s Disease) in order to specify a meta-workflow involving the synchronized and complementary use of CBRAIN, LONI and N4U distributed computing e-infrastructures resources. LINGA intends to execute simultaneously the CIVET cortical thickness extraction workflow on three different but compatible imaging data sets, each of which being hosted by respective infrastructure. The outputs from these three CIVETs are then validated by a quality control process before being grouped and analyzed by a statistical process for large-scale comparison. As far as the N4U infrastructure is concerned, all the tasks were run and monitored using the VirtualLab2DCI API.

The LONI Pipeline Server submit the workflow tasks one by one so the fact the Orchestration service was not yet replaced is not an issue, but, to be able to integrate other tools, the integration of a new WS-BPEL engine is require and is the priority.

3. Testbed provision

As you can see in the deliverable “D7.1 Test-bed installation and API documentation” of the neuGRID project [12], the testbed is composed of the following Grid services:

- **Virtual Organisation Membership Service (VOMS)**: VOMS is a service to manage authorization information in VO scope. The VOMS system should be used to include VO membership and any related authorization information in a user’s proxy certificate. These proxies will be said to have VOMS extensions. The user gives the voms-proxy-init command instead of grid-proxy-init, and a VOMS server will be contacted to check the user’s certificate and create a proxy certificate with VOMS information included. By using that certificate, the VO of a user will be present in every action that he performs.

- **MyProxy (PX)**: On Grids, users authenticate themselves using temporary credentials called proxy certificates, which contain also the corresponding private key. Proxy certificates do not represent a significant security risk, though they are reasonably short-lived (by default, a dozen hours). For longer jobs, PX plays a role of online credential repository [13]. For such long running jobs a proxy renewal system is used, consisting of a Proxy Renewal Service (PRS) on the WMS (see below) and a PX server on a dedicated host. A PX stores long-lived user proxies (with a lifetime of several days, usually) which it uses to generate, on request of the PRS, short-lived proxies for jobs whose proxies are about to expire.

- **LCG File Catalog (LFC)**: It offers a hierarchical view of files to users, with a UNIX-like client interface. The LFC provides Logical File Name (LFN) to Storage URL (SURL) mappings and authorization for file access. The LFNs are aliases created by a user to refer to actual data. Simple metadata can be associated to them. The authorization is performed using UNIX-style permissions, POSIX Access Control Lists (ACL) and VOMS support. The LFC uses a client-server model with a proprietary protocol. LFC server communicates with a database (either Oracle or MySQL), where all the data is stored. LFC catalogue also exposes a Data Location Interface (DLI) - a web service used by applications and Resource Brokers. Provided with a LFN, the DLI returns the actual location of the file replicas.

- **Workload Management System (WMS)**: WMS is a Grid level meta-scheduler that schedules jobs on the available CEs (see below) according to user preferences and several policies. It also consistently keeps track of the jobs it manages

- **Logging and Bookkeeping Service (LB)**: The primary purpose of the Logging and Bookkeeping service is tracking Grid jobs as they are processed by various Grid middleware components. It collects and stores in a database the job status information supplied by the different components of the WMS system. The collection is done by LB local-loggers, which run on the WMS and on the CE, while the LB server, which normally runs on a dedicated server, saves the collected information in the database. The database can be queried by the user from the UI. The information that is gathered in LB is used to inform Grid users on the job state. This information is also useful for example for debugging of user’s jobs.

- **The Information System (IS) (tBDII/sBDII):** The IS provides information about the status of Grid services and available resources. Job and data management services publish their status through Grid Resource Information Server (GRIS).
- **Disk Pool Manager (DPM):** DPM is a service which provides the storage capacity of the Grid middlware. This is a recommended solution for the lightweight deployment of smaller sites because it is easy to install and requires very low maintenance effort. It features full implementation of SRM. Bigger sites usually choose dCache because of robustness, scalability and advanced features. CERN Advanced STORage (CASTOR) is an implementation used by sites that have both disk and tape storage.
- **Computing Element (CE):** CE is the service representing a computing resource. It provides a virtualization of the computing resource localized at a site (typically a batch queue of a cluster but also supercomputers or even single workstations). It provides information about the underlying resource and offers a common interface to submit and manage jobs on the resource.
- **Worker Nodes (WN): WN** is the service which runs on the servers where the jobs are concretely run.

Also, a development **Gateway** and a development **Liferay** instance are deployed in order to test the rest of the developments.

In the next table, you can find a recap of the different services installed with the actualised hostnames:

HOSTNAME/IP	SERVICES
voms.maatg.eu	VOMS
lfc.maatg.eu	LFC
bdii.maatg.eu	tBDII
myproxy.maatg.eu	MyProxy
wms.maatg.eu	WMS
lb.maatg.fr	LB
ng-maat-server3.maatg.eu	DPM
ng-maat-server9.maat-g.com	CREAM CE
site-bdii.maatg.eu	sBDII
ng-maat-server8.maatg.eu	WN
ng-maat-devel1.maatg.fr	Development Gateway for services tests
http://dev.neugrid4you.eu/	Development Liferay instance for Portlet tests

This testbed is fully operational and used on daily bases.

4. Software Release Policy (SRP)

4.1. Overview

Grid-ported algorithms will imply software packages to be managed, installed and configured within the N4U project and its infrastructure. These will represent the core of the N4U platform, which will therefore include the ported algorithms within proper software packages to be deployed, the frontend GUI, all required interface layers between the high-level medical algorithms and the underlying gLite-based Grid middleware layer.

The required packages will be made available from a central project repository, for all site administrators to easily access, and the N4U project will provide lower-level software, in a subsidiary approach to the EGEE/EGI provided tools and packages.

4.2. SRP definition

The N4U platform release policy therefore implements a standard versioning scheme.

New versions of the system are released under a given triplet “x.y.z” of numbers, where “x” indicates the major release number, “y” the minor release number and “z” the bug fix number.

Release 2.0.0, for instance, would be a rigorously tested major release, 2.2.0 would be a minor release based on 2.0.0 and would require less testing, and 2.2.5 would be a bug fix or a “developer'' release based on 2.2.0 and might have undergone virtually no testing at all. Where serious quality is required, 2.0.0 would be the “production'' release. A minor release like 2.2.0 could be flagged as the “current'' version, and users would use it when they are interested in getting newer features and can afford losing some of the stability in exchange. Developers would often work against one of the bug fix, or “test,'' releases such as 2.2.5.

All releases of the system will be made available in the corresponding repositories, whether concerning Development or Production.

Older releases of the system (which are no longer in use in the concerned environments), will be kept in the releases repository for about a month, and then be archived.

4.3. SRP framework

N4U technical partners setup a central software repository for releasing new versions of the platform (http://repo.maatg.fr/apt/pandora). The latter will be used by Development and Production site administrators, as a reference repository for future installations.

5. Conclusions

This document aims at provide the N4U application developers with a documentation about the DCI Pipeline APIs, the testbed and the SRP that was put in place.

The two APIs has been described: ExpressLane2DCI and VirtualLab2DCI.

In the ExpressLane2DCI API section, a full description of the concepts and of the usage has been provided. This API is at the time of writing highly tested in order to have it stable really quickly.

In the VirtualLab2DCI API section, a description of the current architecture was given. This later is the result of the work that has been done in the neuGRID project and also from the migration to the Apache CXF Service Framework that has been done. As it was explained, the current architecture has some limitations that will be corrected soon, by integrated a WS-BPEL workflow engine. The current implementation of the Pipeline service is in place and is already connected with the LONI Pipeline server using a specific plugin that was created. Everything was validated by the LINGA data challenge.

As far as the ExpressLane2DCI API is concerned, we are close the final version and only minor bug fixes should occurs. As far as the VirtualLab2DCI API is concerned, the priority now is to integrate a WS-BPEL engine which will provide the task “orchestration” functionality. Having such a component will provide the applications which will use the system with the ability to orchestrated pure grid tasks but also web services.

The second part of the document quickly presented the testbed that was put in place. This testbed, which is the practically the same than the neuGRID project one with some adaptations, is fully operational and used on daily bases.

The latest part presented the SRP that is followed in the project and which help to provide a stable production environement.

6. Bibliography

[1] The Simple API for Grid Applications workgroup website - https://forge.ogf.org/sf/projects/saga-rg

[2] The OASIS Web Services Business Process Execution Language (WSBPEL) - https://www.oasis-open.org/committees/wsbpel/

[3] The JSAGA project website - http://grid.in2p3.fr/jsaga/

[4] neuGRID WP10 deliverable D10.1 – "Gridification Model Specification"

[5] neuGRID WP6 deliverable D6.2 – "Interim service prototype report"

[6] Grimoires: A UDDI-compatible Web Service Registry with Metadata Annotation Extensions - http://code.google.com/p/grimoires/

[7] The Crux project Presentation - http://confluence.globus.org/display/whi/Crux+for+GT+Developers

[8] Apache CXF: An Open-Source Services Framework - http://cxf.apache.org/

[9] Online Condor DAGMan Users' Manual: Explanations about DAG (directed acyclic graph) jobs - http://research.cs.wisc.edu/condor/manual/v6.6/2_11DAGMan_Applications.html

[10] The CREAM wiki website - https://wiki.italiangrid.it/twiki/bin/view/CREAM/WebHome

[11] N4U WP8 deliverable D8.2 – "N4U Science Gateway Release Report"

[12] neuGRID WP7 deliverable D7.1 – "Test-bed installation and API documentation"

[13] MyProxy - http://grid.ncsa.uiuc.edu/myproxy/

[14] The LONI Pipeline website - http://pipeline.loni.ucla.edu/

[15] Haring Interoperable Workflows for large-scale scientific simulations on Available DCIs (SHIWA) - FP7 Capacities Programme under contract number RI-261585-WP4 deliverable D4.4 – "Sub-contracted applications implementation"

7. Annex 1

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="Pipeline" targetNamespace="http://pandora.maatg.fr/ns/sal"
             xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:sal="http://pandora.maatg.fr/ns/sal"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xmime="http://www.w3.org/2005/05/xmlmime">

   <!--============================================================ -->
   <!-- T Y P E S -->
   <!--============================================================ -->
   <wsdl:types>
      <xsd:schema attributeFormDefault="qualified" elementFormDefault="qualified"
         targetNamespace="http://pandora.maatg.fr/ns/sal">
                              <!-- ######################################################## -->
                              <!-- ############## DATAMODEL REPRESENTATION ############## -->
                              <!-- ######################################################## -->
                              <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
                        <xsd:complexType name="Activity" abstract="true">
                           <xsd:sequence>
                              <xsd:element name="ActivityID" type="xsd:string" nillable="false"/>
                              <xsd:element name="Arguments" type="sal:Argument" minOccurs="0" maxOccurs="unbounded" />
                              <xsd:element name="Inputs" type="sal:WorkflowAttachedFile" minOccurs="0" maxOccurs="unbounded" />
                              <xsd:element name="Outputs" type="sal:WorkflowAttachedFile" minOccurs="0" maxOccurs="unbounded" />
                           </xsd:sequence>
                        </xsd:complexType>
                        <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
                              <xsd:complexType name="Task">
                                    <xsd:complexContent>
                                          <xsd:extension base="sal:Activity">
                                                <xsd:sequence>
                                                      <xsd:element name="Executable" type="xsd:string"
nillable="false" />
                              <xsd:element name="Requirements" type="xsd:string" nillable="true" />
                              <xsd:element name="JobType" type="xsd:string" nillable="true" />
                                                </xsd:sequence>
                                          </xsd:extension>
                                    </xsd:complexContent>
                              </xsd:complexType>
                        <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
                        <xsd:complexType name="WebService">
                           <xsd:complexContent>
                              <xsd:extension base="sal:Activity">
                                 <xsd:sequence>
                                    <xsd:element name="EndpointURL" type="xsd:string" nillable="false" />
                                    <xsd:element name="FunctionName" type="xsd:string" nillable="false" />
                                 </xsd:sequence>
                              </xsd:extension>
                           </xsd:complexContent>
                        </xsd:complexType>
                        <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
                        <xsd:complexType name="Argument">
                           <xsd:sequence>
                              <xsd:element name="Name" type="xsd:string" nillable="true"/>
                              <xsd:element name="Value" type="xsd:string" nillable="false"/>
                           </xsd:sequence>
                        </xsd:complexType>
                        <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
                        <xsd:complexType name="Dependency">
                           <xsd:sequence>
                              <xsd:element name="ActivityID" type="xsd:string" nillable="false"/>
                              <xsd:element name="Dependencies" type="xsd:string" minOccurs="1" maxOccurs="unbounded" />
                           </xsd:sequence>
                        </xsd:complexType>
                        <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
                        <xsd:complexType name="Workflow">
                           <xsd:sequence>
                              <xsd:element name="WorkflowID" type="xsd:string" nillable="true"/>
                              <xsd:element name="Activities" type="sal:Activity" minOccurs="1" maxOccurs="unbounded" />
                              <xsd:element name="Dependencies" type="sal:Dependency" minOccurs="0" maxOccurs="unbounded" />
                           </xsd:sequence>
                        </xsd:complexType>
                              <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
                              <xsd:simpleType name="WorkflowStatusState">
                                    <xsd:restriction base="xsd:string">
                                          <xsd:enumeration value="SUBMITTED" />
                                          <xsd:enumeration value="QUEUED" />
                                          <xsd:enumeration value="RUNNING" />
                                          <xsd:enumeration value="DONE" />
                                          <xsd:enumeration value="CLEARED" />
                                          <xsd:enumeration value="CANCELED" />
                                          <xsd:enumeration value="FAILED_ERROR" />
                                          <xsd:enumeration value="FAILED_ABORTED" />
                                          <xsd:enumeration value="SUSPENDED_QUEUED" />
                                          <xsd:enumeration value="SUSPENDED_ACTIVE" />
                                          <xsd:enumeration value="UNKNOWN" />
                                    </xsd:restriction>
```

```
                                </xsd:simpleType>
                                <xsd:complexType name="WorkflowStatus">
                                        <xsd:sequence>
                                                <xsd:element name="WorkflowID" type="xsd:string" nillable="false" />
                                                <xsd:element name="State" type="sal:WorkflowStatusState"/>
                                                <xsd:element name="ExitCode" type="xsd:int" nillable="true" />
                                                <xsd:element name="Reason" type="xsd:string" nillable="true" />
                                                <xsd:element name="startDate" type="xsd:dateTime" nillable="true" />
                                                <xsd:element name="endDate" type="xsd:dateTime" nillable="true" />
                                        </xsd:sequence>
                                </xsd:complexType>
                                <xsd:complexType name="WorkflowStatuses">
                                        <xsd:sequence>
                                                <xsd:element name="WorkflowStatuses" type="sal:WorkflowStatus" minOccurs="0"
maxOccurs="unbounded"/>
                                        </xsd:sequence>
                                </xsd:complexType>
                        <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
                        <xsd:complexType name="CancelWorkflow">
                           <xsd:sequence>
                              <xsd:element name="WorkflowID" type="xsd:string" nillable="false"/>
                              <xsd:element name="Force" type="xsd:boolean" />
                           </xsd:sequence>
                        </xsd:complexType>
                        <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
                        <xsd:complexType name="WorkflowOutput">
                           <xsd:sequence>
                              <xsd:element name="WorkflowID" type="xsd:string" nillable="false"/>
                              <xsd:element name="OutputFiles" type="sal:WorkflowAttachedFile" minOccurs="0" maxOccurs="unbounded"/>
                           </xsd:sequence>
                        </xsd:complexType>
                        <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
                        <xsd:complexType name="WorkflowAttachedFile">
                           <xsd:sequence>
                              <xsd:element name="Name" type="xsd:string" nillable="false"/>
                              <xsd:element name="Path" type="xsd:string" nillable="true"/>
                              <xsd:element name="Content" type="xsd:base64Binary" xmime:expectedContentTypes="application/octet-stream"
nillable="true"/>
                           </xsd:sequence>
                        </xsd:complexType>


                        <!-- ########################################################## -->
                        <!-- ################## SUBMIT FUNCTION ###################### -->
                        <!-- ########################################################## -->
                        <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
                        <xsd:element name="workflowSubmit" type="sal:Workflow" />
                        <xsd:element name="workflowSubmitResponse" type="xsd:string" />

                        <!-- ########################################################## -->
                        <!-- ################## CONTROL FUNCTIONS ##################### -->
                        <!-- ########################################################## -->
                        <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
                        <xsd:element name="listWorkflows" type="xsd:string" />
                        <xsd:element name="listWorkflowsResponse" type="sal:WorkflowStatuses" />
                        <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
                        <xsd:element name="cancelWorkflow" type="sal:CancelWorkflow" />
                        <xsd:element name="cancelWorkflowResponse" type="xsd:boolean" />
                        <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
                        <xsd:element name="getWorkflowStatus" type="xsd:string" />
                        <xsd:element name="getWorkflowStatusResponse" type="sal:WorkflowStatus" />
                        <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
                        <xsd:element name="getWorkflowOutput" type="xsd:string" />
                        <xsd:element name="getWorkflowOutputResponse" type="sal:WorkflowOutput" />
                        <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
                        <xsd:element name="purgeWorkflow" type="xsd:string" />
                        <xsd:element name="purgeWorkflowResponse" type="xsd:boolean" />

        </xsd:schema>
    </wsdl:types>

    <!--============================================================ -->
    <!-- M E S S A G E S-->
    <!--============================================================ -->

    <!-- ################### SUBMIT MESSAGE ################### -->
    <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
    <message name="workflowSubmitInputMessage">
        <part name="parameters" element="sal:workflowSubmit"/>
    </message>
    <message name="workflowSubmitOutputMessage">
        <part name="parameters" element="sal:workflowSubmitResponse"/>
    </message>

    <!-- ################### CONTROL MESSAGES ################### -->
    <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
    <message name="listWorkflowsInputMessage">
        <part name="parameters" element="sal:listWorkflows"/>
```

```
</message>
<message name="listWorkflowsOutputMessage">
   <part name="parameters" element="sal:listWorkflowsResponse"/>
</message>
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<message name="cancelWorkflowInputMessage">
   <part name="parameters" element="sal:cancelWorkflow"/>
</message>
<message name="cancelWorkflowOutputMessage">
   <part name="parameters" element="sal:cancelWorkflowResponse"/>
</message>
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<message name="getWorkflowStatusInputMessage">
   <part name="parameters" element="sal:getWorkflowStatus"/>
</message>
<message name="getWorkflowStatusOutputMessage">
   <part name="parameters" element="sal:getWorkflowStatusResponse"/>
</message>
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<message name="getWorkflowOutputInputMessage">
   <part name="parameters" element="sal:getWorkflowOutput"/>
</message>
<message name="getWorkflowOutputOutputMessage">
   <part name="parameters" element="sal:getWorkflowOutputResponse"/>
</message>
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<message name="purgeWorkflowInputMessage">
   <part name="parameters" element="sal:purgeWorkflow"/>
</message>
<message name="purgeWorkflowOutputMessage">
   <part name="parameters" element="sal:purgeWorkflowResponse"/>
</message>


<!--============================================================ -->
<!-- P O R T T Y P E S  -->
<!--============================================================ -->
<portType name="PipelineServicePortType">

   <!-- ################### SUBMIT OPERATION ################# -->
   <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
   <operation name="workflowSubmit">
      <input message="sal:workflowSubmitInputMessage"/>
      <output message="sal:workflowSubmitOutputMessage"/>
   </operation>

   <!-- ################## CONTROL OPERATIONS ################## -->
   <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
   <operation name="listWorkflows">
      <input message="sal:listWorkflowsInputMessage"/>
      <output message="sal:listWorkflowsOutputMessage"/>
   </operation>
   <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
   <operation name="cancelWorkflow">
      <input message="sal:cancelWorkflowInputMessage"/>
      <output message="sal:cancelWorkflowOutputMessage"/>
   </operation>
   <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
   <operation name="getWorkflowStatus">
      <input message="sal:getWorkflowStatusInputMessage"/>
      <output message="sal:getWorkflowStatusOutputMessage"/>
   </operation>
   <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~  -->
   <operation name="getWorkflowOutput">
      <input message="sal:getWorkflowOutputInputMessage"/>
      <output message="sal:getWorkflowOutputOutputMessage"/>
   </operation>
   <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
   <operation name="purgeWorkflow">
      <input message="sal:purgeWorkflowInputMessage"/>
      <output message="sal:purgeWorkflowOutputMessage"/>
   </operation>
</portType>
<!--============================================================ -->
         <!-- B I N D I N G -->
         <!--============================================================ -->
         <binding name="PipelineServiceBinding" type="sal:PipelineServicePortType">
                  <soap:binding style="document"
                           transport="http://schemas.xmlsoap.org/soap/http" />
                  <operation name="workflowSubmit">
                           <soap:operation soapAction="" />
                           <input>
                                    <soap:body use="literal" />
                           </input>
                           <output>
                                    <soap:body use="literal" />
                           </output>
                  </operation>
```

```
		<operation name="listWorkflows">
			<soap:operation soapAction="" />
			<input>
				<soap:body use="literal" />
			</input>
			<output>
				<soap:body use="literal" />
			</output>
		</operation>
		<operation name="cancelWorkflow">
			<soap:operation soapAction="" />
			<input>
				<soap:body use="literal" />
			</input>
			<output>
				<soap:body use="literal" />
			</output>
		</operation>
		<operation name="getWorkflowStatus">
			<soap:operation soapAction="" />
			<input>
				<soap:body use="literal" />
			</input>
			<output>
				<soap:body use="literal" />
			</output>
		</operation>
		<operation name="getWorkflowOutput">
			<soap:operation soapAction="" />
			<input>
				<soap:body use="literal" />
			</input>
			<output>
				<soap:body use="literal" />
			</output>
		</operation>
		<operation name="purgeWorkflow">
			<soap:operation soapAction="" />
			<input>
				<soap:body use="literal" />
			</input>
			<output>
				<soap:body use="literal" />
			</output>
		</operation>
	</binding>

	<!--============================================================ -->
	<!-- S E R V I C E -->
	<!--============================================================ -->
	<service name="PipelineService">
		<port name="PipelineServicePort" binding="sal:PipelineServiceBinding">
			<soap:address
				location="https://localhost:8443/pandora-gateway-sal-pipeline/pipeline" />
			<wsp:PolicyReference URI="#Policy"
				xmlns:wsp="http://www.w3.org/ns/ws-policy" />
		</port>
	</service>

	<!--============================================================ -->
	<!-- P O L I C Y -->
	<!--============================================================ -->
	<wsp:Policy wsu:Id="Policy" xmlns:wsp="http://www.w3.org/ns/ws-policy"
		xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
		xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
		<wsp:ExactlyOne>
			<wsp:All>
				<!-- TLS with client certificate -->
				<sp:TransportBinding>
					<wsp:Policy>
						<sp:TransportToken>
							<wsp:Policy>
								<sp:HttpsToken RequireClientCertificate="true" />
							</wsp:Policy>
						</sp:TransportToken>
					</wsp:Policy>
				</sp:TransportBinding>
			</wsp:All>
		</wsp:ExactlyOne>
	</wsp:Policy>
</definitions>
```