Grant agreement no.211714

neuGrid

A GRID-BASED e-INFRASTRUCTURE FOR DATA ARCHIVING/ COMMUNICATION AND COMPUTATIONALLY INTENSIVE APPLICATIONS IN THE MEDICAL SCIENCES

Combination of Collaborative Project and Coordination and Support Action

Objective INFRA-2007-1.2.2 - Deployment of e-Infrastructures for scientific communities

Deliverable reference number and title: **D6.1 Distributed Medical Services Provision (Pipeline Service)**

Due date of deliverable: **Month 12**

Actual submission date: **31st January 2009**

Start date of project: **February 1st 2008** Duration: **36 months**

Organisation name of lead contractor for this deliverable: **University of the West of England, Bristol UK**

Revision: Version **1**

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)		
Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Table of Contents

Intended Recipients

The WP6 workpackage entitled **"Distributed Medical Services Provision"** aims to design a group of *generic* services that can be used in a number of related medical applications. These will then be implemented in order to fulfil the neuGrid specific project requirements. The services will be built according to the design philosophy presented in the WP6 deliverable. This will help to enhance and promote their re-usability in other related applications.

This deliverable document presents a design philosophy that the generic services will follow, maps user requirements against suitable services and briefly presents a list of the services. An initial implementation of the services and their detailed API descriptions will be delivered in the year 2 deliverable.

The WP leaders, technical users and neuGrid developers are the intended recipients of this document. To a lesser extent, since indirectly concerned (through the natural abstraction of Workflow/ Pipeline authoring environments such as the ones proposed in WP6), neuro-scientists and prospective users (e.g. Pharmaceutical companies) as well as internal and external reviewers of the project activities, are anticipated as potential readers of this document.

6. The Pipeline Service

Neuro-imaging pipelines allow neuroscientists and clinicians to apply series of automated transformations and processes on brain images for decision support purposes using complex and nested workflows. Often these processes are very compute intensive and deal with large amounts of data. Grid enabled neuro-imaging pipeline services are either proprietary or under research and neuroscientists have to rely on command line scripts to design and execute the pipelines. The role of the Pipeline Service is to enable scientists to create and design workflows in a user-friendly fashion, to grid-enable and to enact these pipelines over a Grid, and finally allow users to view the results of the execution. In neuGrid, the fundamental functionality of the Pipeline Service includes the following:

- Enable the authoring of the pipeline in a user friendly environment, using the Neuralyze executables as actors of the pipelines;
- Parallelize and Grid-enable the abstract user defined pipeline for optimal execution over a grid;
- Submit and enact the pipeline for execution on Grid and
- View results of the execution as well as intermediary provenance data.

Processing pipelines (as shown in Figure 10) are compound jobs composed of several atomic stages (each stage being an algorithm applied to an input dataset and producing an output dataset). Each of these stages can be processed on different machines. Stages are chained (eg the output of one stage is used as input for the next stage) but are not necessarily in series (stages can be processed in parallel). Therefore, the Pipeline Service should offer a graphical pipeline description mechanism to draw the structure of pipelines and a smart scheduler able to exploit the pipeline's intrinsic parallelism by distributing processing on various Grid nodes (data-flow control, load balancing, synchronization etc). Pipelines are of real interest when processing a large number of input data rather than a single input. Through pipelines, the user can describe once and for all the chain of transformations that each element of the input dataset should undergo. The pipeline scheduler can process several elements in parallel on Grid nodes (thousands of concurrent input images are expected for some medical applications). Synchronization barriers may be needed to extract statistics from processed data at some point(s) in the process flow. Therefore, pipelines should provide additional services such as synchronization and provenance, logs of accomplished stages for a given input, restart from a failed job, automatic resubmission of stages that failed for user-independent reasons, etc. The following diagram shows a scenario where a pipeline is created and executed on a set of images.

The design of the Pipeline Service is dictated according to the design philosophy and constrained by the neuGrid user requirements. All neuGrid specific user requirements as well as technical requirements, evaluations of state of the art Grid pipeline services as well as design of the neuGrid Pipeline Service are detailed in the Pipeline Service design document.

6.1 Introduction

The neuGrid generic medical services layer includes numerous components that facilitate the execution of a neuro-imaging pipeline on a grid infrastructure. One of the central services enabling this is the Pipeline Service. The functionality of the Pipeline Service is mandated by specific requirements from WP6 and WP10.

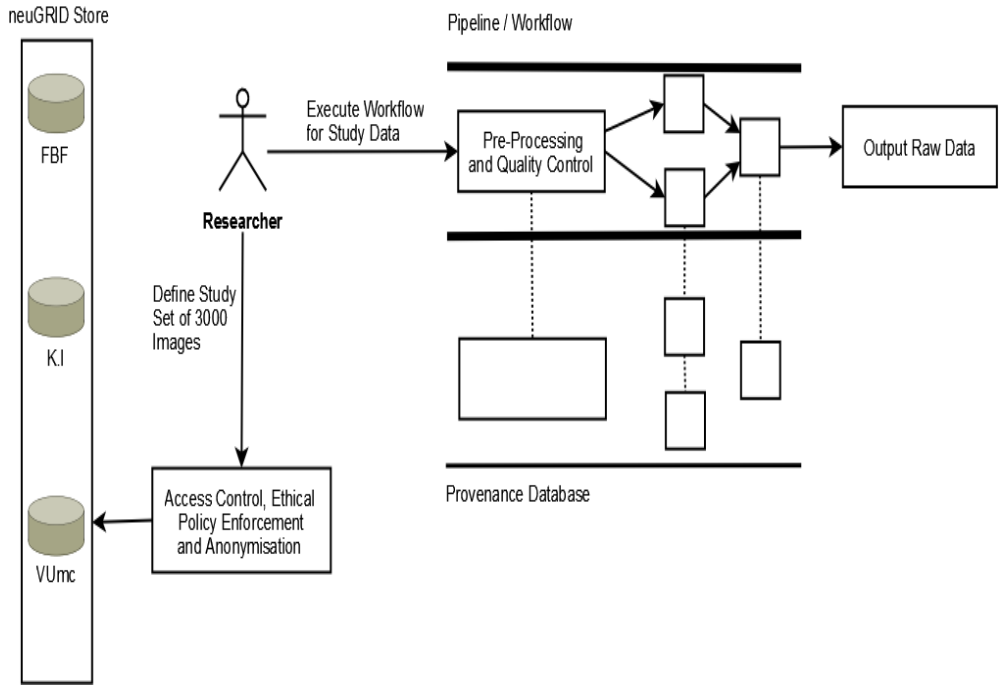

Figure 10: A conceptual model of a neuro-imaging Pipeline

The role of the Pipeline Service, as outlined earlier, is to enable scientists to create and design workflows in a user-friendly fashion, grid-enable and enact the pipeline over a grid, and finally allow users to view the results of the execution. The fundamental components of the Pipeline Service are depicted in Figure 11.

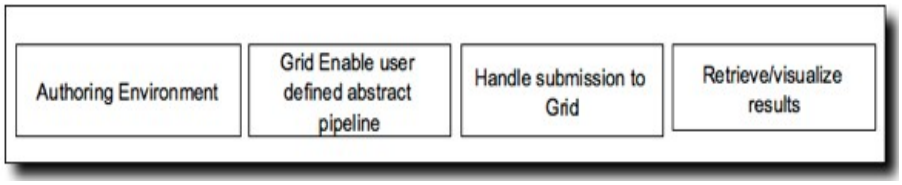

Figure 11: Components of a Pipeline Service

The WP6 Design Philosophy mandates that generic medical neuGrid services should be developed on SOA principles. The SOA implementation adapted for neuGrid is the Web Services Stack. Hence the Pipeline Service should be developed as a web service supporting a user-friendly pipeline authoring front end. At the back end, the Pipeline Service should seamlessly interoperate with various other WP6 services.

Web Services are built via standardized environments and interfaces. Because of standardized interfaces, there is a potential for re-usability of the services in various configurations to enable things for which the initial architecture was not designed to support. For example, neuGrid requires a Pipeline Service, which can support NE actors and other neuro-imaging algorithms; however the same service can be used for any other application in any environment as long as the standard interfaces are used. The architecture itself is flexible, new services can be developed in future and embedded into the architecture with minimal or no modification to any other services, as they are loosely coupled. Finally because of standardized interfaces the internals of the service can be developed on any platform, providing a platform independent solution.

The purpose of this document is to present a design of the Pipeline Service. The design is guided by the relevant user requirements and the evaluations of the state-of-the-art technologies. The document proceeds as follows: section 6.2 outlines relevant neuGrid user requirements, which will govern the design of the Pipeline Service. Since a Pipeline Service can be divided into these constituent parts: Authoring environment, grid-enabling and enactment mechanism, the associated state-of-the-art projects are presented accordingly, in section 6.3, 6.4 and section 6.5 respectively. Section 6.6 presents tentative architectures of the Pipeline Service, outlining advantages and disadvantages of each. Recommendations on the architecture are presented in section 6.7. Section 6.8 presents the detailed end to end design of the Pipeline Service, using the components, which most closely match user requirements. Section 6.9 presents research issues and potential future work, which will be carried out as part of the Pipeline Service development.

The words pipeline and workflows are used interchangeably. A distinction in the document is made between task-based workflows and services-based workflows. Task based workflows are workflows which constitute of actors which are simple executable processes. Service based workflows on the other hand, consist of workflows, which carry out interactions amongst web services to complete a workflow.

6.2 Pipeline Service User Requirements

neuGrid Pipeline Service has four primary components: Authoring, workflow planning, workflow enactment and results retrieval and viewing. There are specific user requirements for each component of the Pipeline Service. This section highlights the specific relevant user requirements. It is necessary for any component of the Pipeline Service to be compliant with the user requirements as they will be the primary means of judging suitability of software components. These requirements will also guide the evaluation of the state-of-the-art technologies.

The neuGrid user requirements are drawn from consultations with the potential users of the system, and are not final. Requirements will evolve as the project proceeds. The stated requirements in this section are written from a technical standpoint as opposed to the requirements document, which documents requirements from a user perspective.

Following the separation of roles of the Pipeline Service in Figure 11, the requirements are presented in a similar fashion.

6.2.1. Workflow Authoring

Workflow authoring is one of the essential components of neuGrid Pipeline Service. Workflow authoring is further divided into: construction, editing, validation, annotation and visualization of workflows. The following are the essential requirements under this category, as depicted in Figure 12. These requirements are derived from the neuGrid User Requirements document. Each category is detailed in subsequent sections.

6.2.1.1. Construction

Constructing pipelines is a fundamental task of the Pipeline Service. As per user requirements, constructing a pipeline should involve the creation and editing of workflows by using a graphical user interface (GUI). The GUI should support the construction of the pipeline by dragging and dropping neuro-imaging algorithms, and stringing their outputs and inputs together to create workflow.

The authoring environment should represent neuro-imaging algorithms as modules. It should be possible for users to query for a particular module and use it in their pipeline, by dragging it form

the "toolbox" and dropping it in the pipeline authoring workspace. Users should be able to specify input and output arguments of the module in the workspace. The environment should enable the user to connect the output of a module to the input of a subsequent module. This connection should be specified in a visual manner (i.e. connect the output of a module with a line to the input of another).

Validation of constructed pipeline is also important during authoring. Before submitting a workflow to the Grid, users should be able to check the validity of the pipeline from the authoring environment. Validation involves checking if the output of a module is of the required format for an input for another module. For example, if the input of a module requires numeric parameters, it should not be connected to a DICOM image file input.

In order to facilitate construction of pipelines users should be allowed to string together complete pipelines. Users do not want to be restricted to authoring pipelines where they have to construct a pipeline by specifying module after module. Rather the authoring environment should be flexible enough to enable the connection of entire pipelines. Selecting algorithms for authoring is a related requirement. Users want a panel view which should list existing modules, and user should be able to drag and drop those modules into their workspace, at the same time, users want to be able to search and query for existing modules, in order to facilitate quick usage.

6.2.1.2 Editing Workflows

Editing already constructed workflow should be done in a similar fashion as constructing new workflows. Editing workflows may involve operations such as changing input parameters to certain modules or adding new modules in the pipeline.

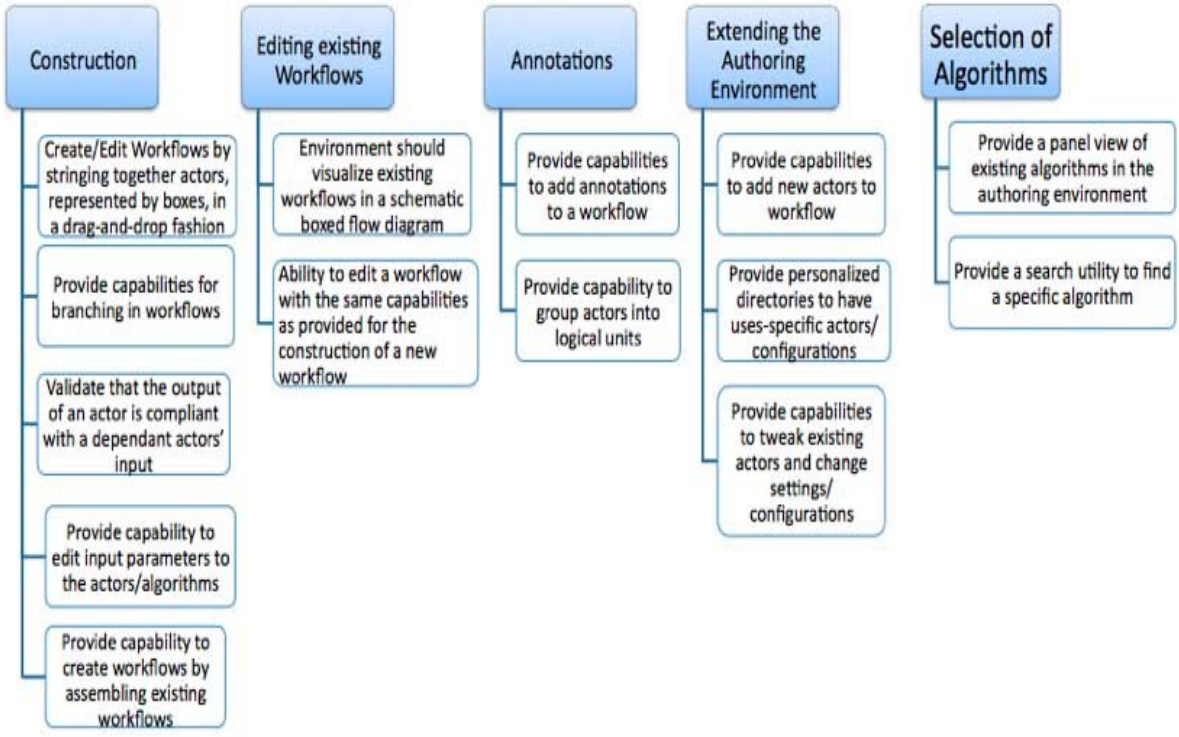

Figure 12: Workflow authoring related requirements

6.2.1.3 Annotating Workflows

Annotations to workflows are important to the users of neuGrid. Users want to author pipelines, document annotation on top of the pipelines, cluster modules together to form logical blocks, as well as add general comments for themselves or other users. All these facilities should be provided by the workflow authoring environment.

6.2.1.4 Extending the authoring Environment

neuGrid services are designed to be extensible and evolvable. The user workflow-authoring environment will cover a wide range of modules. Requirements for new modules may however arise as the system evolves. Hence users require an easy to use mechanism to add new modules. Another crucial requirements is the user-specific configurability of the environment. Users may want to modify the look and feel of the authoring environment to suite their use, as well as save workflows/pipeline specific settings to their own custom user specific directory/storage/repository.

6.2.2 Workflow Optimization

There are few user requirements about workflow optimization, as depicted in Figure 13. The requirements for pipeline optimization are derived from specific tasks from the neuGrid Project Proposal. Most of the requirements are from WP10, which covers the gridification and optimization of existing pipelines.

Figure 13: Planning and Optimization Requirement

Because neuGrid constitutes a Grid of multiple sites, not all sites may have sufficient bandwidth in order to enable seamless data sharing. Hence it is important for the pipeline optimization mechanism to take data locality into considerations, if a site is executing a pipeline and the data is available in the local site, the pipeline execution should stay local.

These set of requirements, depicted in Figure 14, deal with the control over execution of workflows, as well as the viewing of results and errors in case they arise.

6.2.3 Workflow Execution and Results

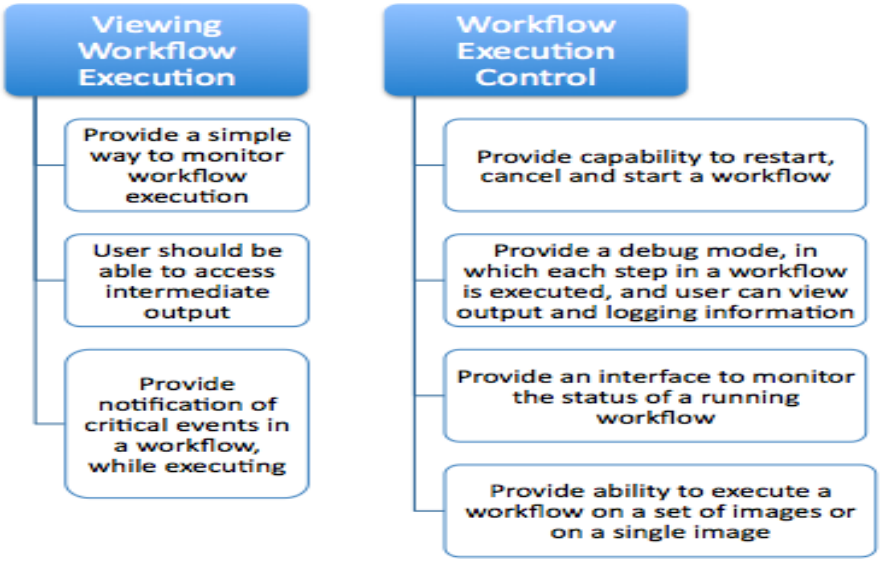

Figure 14: Workflow Execution and Results

6.2.3.1 Workflow Execution and Control

After authoring a pipeline users want to submit it for execution. The Pipeline Service does the appropriate workflow planning and optimization and starts enacting it to execute the pipeline. To control the enactment and execution process users want various control functionality from the Pipeline Service. Such control functionality includes the ability to restart the execution of a pipeline and terminate it before completion.

An important requirement is to be able to execute a workflow step by step, in a debug mode, and view intermediate data outputs and logs. Users also want monitoring information to be displayed, and be notified of critical events as they arise.

The Pipeline Service should allow users to execute a workflow and process a complete set of images, or execute it and process a single image.

6.2.3.2 Viewing Workflow Execution

neuGrid users have also specified specific requirements on the ability to view the output of a pipeline. Users want up-to-date monitoring information, as well as seamless access to the intermediate data produced while a pipeline is executing. Users also want notifications of critical events.

6.2.3 Criteria to evaluate related projects

The documented user requirements define the role of the neuGrid Pipeline Service. The following classification, as presented in Table 1, which is derived from the user requirements, will be used to carry out the evaluation of related state-of-the-art projects.

6.3 Existing Pipeline Authoring and Execution Environments

This section outlines related evaluated projects. The purpose of this section is to survey and explore what relevant solutions are already available and if any of the existing solutions satisfies

the user requirements. Some of the solutions may be open source; hence if they satisfy a subset of the requirements they can be selected for extension in order to inculcate support for all requirements. After the description of projects their evaluation with respect to the user requirements is presented.

Easy Construction of Workflow
Ability to easily edit existing workflows
Annotation capabilities
Simple mechanism for adding new modules
Ability to query and select algorithms
Optimize and Grid-Enable Algorithms
Ability to view results and associated Monitoring Data
Control Execution of a Workflow

Table 1: Evaluation criteria

6.3.1 LONI Pipeline

LONI Pipeline [30] provides a rich interface for creating and enacting neuro-imaging pipelines (see Figure 15). It supports the creation of pipelines using popular neuro-imaging algorithms such as MNI MINC, AIR, Brainsuite and others. LONI Pipelines' authoring environment supports the creation of task-based workflows via a user-friendly drag and drop interface. It does not support staging of workflow actors, rather they must be pre-installed on the servers where execution takes place. Enactment of LONI pipelines is carried out on local servers where the executable processes are deployed. LONI Pipeline also supports enactment over DRMAA to a compliant cluster execution service such as Sun Grid Engine [33]. LONI also enables the users to keep track of intermediary output and the execution states of actors during execution.

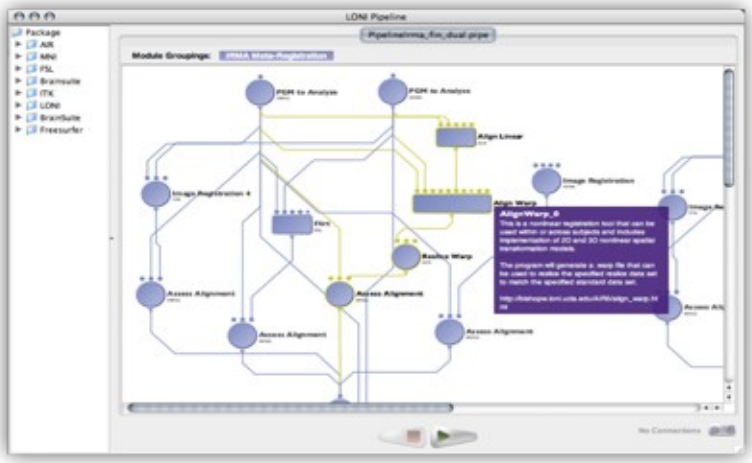

Figure 15: LONI Pipeline Interface

Because LONI Pipeline is a complete neuro-imaging toolkit a detailed description of LONI is provided in Appendix A. Major drawbacks of LONI include:

- LONI Pipeline is a proprietary tool.
- The LONI Pipeline Server module, which is the primary module orchestrating pipeline execution, is limited in capabilities.
    - It does not support interoperability to Grid infrastructures; however DRMAA access is provided to selected cluster execution services.
    - Because basic Grid support is missing, LONI is not capable of dealing with large-scale data processing and analysis.

6.3.2. Taverna

The Taverna Workbench provides a desktop authoring environment and enactment engine for scientific workflows expressed in the Simple Conceptual Unified Flow language (SCUFL) (see Figure 16). Taverna provides a desktop authoring environment where workflows are designed and created. Because Taverna is an authoring environment for SCUFL workflows, it is restricted to the authoring of only service-based workflows. Enactment of workflows in the Taverna workbench is done via the built-in Freeflou enactment engine. Provenance data is also displayed which constitutes of the intermediary results as well as invocation made to constituent web services and the responses returned. Taverna allows the inclusion of local widgets such as Java classes; beans shell scripts and others, which allow for minor operations to be carried out on responses from SOAP messages before other services are invoked.

Taverna has a rich interface. It supports extensions via plug-ins, which for example give it remote execution capability. It exports an API, which allows the invocation of workflows from scripts without any interface interaction. This makes it possible to create front ends, like AJAX based frontends, which enable the invocation of workflows from simple web browsers.

6.3.3 Kepler

Kepler [15] is a software application for the analysis and modelling of scientific data. Kepler is a generic scientific workflow environment. Kepler is built around on a ‘director’ concept. Each director customizes the way Kepler executes a workflow, for instance if workflows are created with the Synchronous Dataflow Director (SDF), the execution of the workflow will be done locally in a sequential fashion, however if the same workflow is created via the Process Network Director, Kepler will try to execute the workflow in a more parallel fashion. This director concept allows Kepler to be a generic tool that can be customized to any environment. It is possible to create service-based workflows (as depicted in Figure 17) as well as task-based workflows. For Grids, a director for Pegasus [20] has been developed. This director executes a Kepler created workflow on a grid infrastructure after having it grid-enabled via Pegasus (see section 6.4.2). Kepler is extensible; users can develop and add new directors to customize the behaviour of the environment.

In terms of authoring capabilities, Kepler supports drag and drop workflow construction and annotations. The interface is shown in Figure 17.

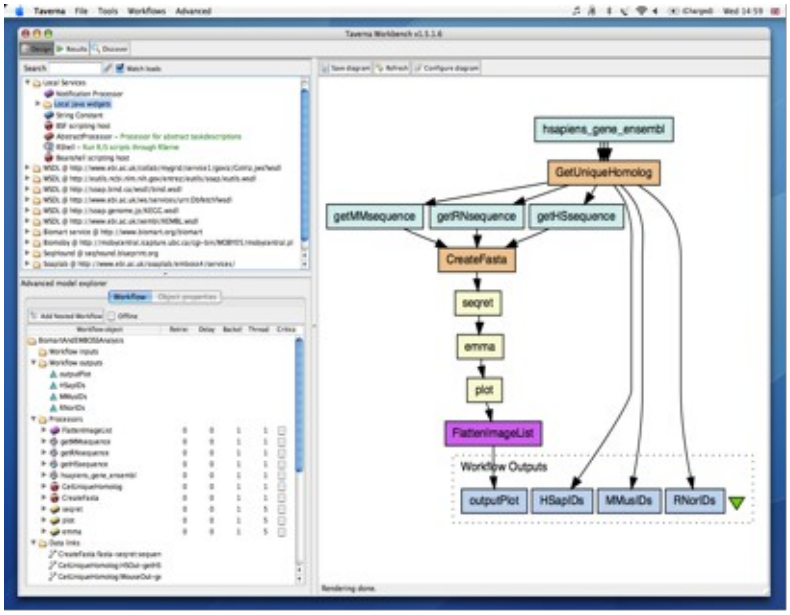

Figure 16: Taverna Interface

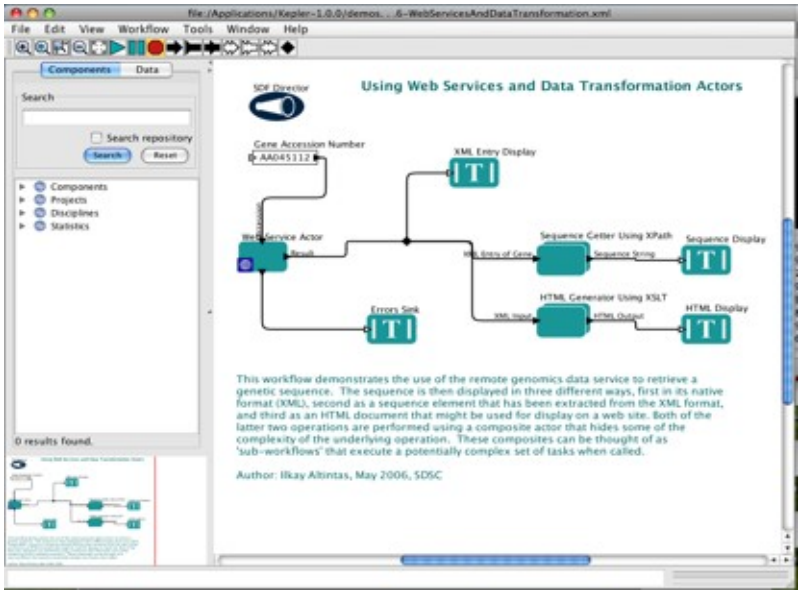

Figure 17: Kepler

6.3.4 Triana

Triana [27] is a graphical workflow environment, which allows the authoring of both task based and service based workflows. Triana was developed for the GEO600 gravitational wave experiment. Triana is primarily used to analyse terabytes of data generated in the GEO600 experiment. Triana is designed to make is it possible for scientists in the project to examine this data in a simple and versatile way. It includes various out-of –the-box toolboxes which include tasks for signal-analysis toolkit, an image-manipulation toolkit, a desktop publishing toolkit, and others.

Triana supports both service-based workflows and task-based workflows. For enacting service-based workflow Triana uses a Grid Application Prototype interface. For task based workflow enactment the Gridlab GAT API [14] is used. Both these interfaces have multiple bindings, which allow different service/grid middleware to be employed without amending the Triana application code. For service-oriented components, web services and P2PS services can currently be invoked, while for grid-oriented components, job submission can currently be done using GRMS, GRAM or the local Fork adaptor. Both the GAP Interface and the Gridlab GAT allow new bindings to be plugged in when they become available.

6.3.5. Poor Man's Pipeline

The Poor Man's Pipeline (PMP) is a simple PERL module based pipelining environment. It is the primary pipeline tool used for NE pipelines. PMP offers a programmatic way to construct a pipeline. It provides an API through which users describe stages of the pipeline. Descriptions include input, output data sets as well as dependencies of pipeline actors. Parallel execution based on data parallelism is supported in PMP. PMP supports the execution of pipelines over the Sun Grid Engine and PBS.

6.3.6. Matching of User Requirements

The following table evaluate all the individual workflow environments against relevant user requirements evaluation criteria detailed in section 6.6.2. The super-scripted numbers are explained after the table.

\User Requirements	LONI Pipeline	Kepler	Taverna	Triana	PMP
Easy Construction of Workflow	Supported	Supported	Supported	Supported	Not Supported[1]
Ability to easily edit existing workflows	Supported	Supported	Supported	Supported	Not Supported[2]
Annotation capabilities	Supported	Supported	Supported	Supported	Not Supported[3]
Simple mechanism for adding new modules	Supported	Supported	Supported	Supported	Supported
Ability to query and select algorithms	Supported	Supported	Supported	Supported	Not Supported[4]
Optimize and Grid-Enable Algorithms	Not Supported[5]	Not Supported[6]	Not Supported[7]	Not Supported[8]	Not Supported[9]
Ability to view results and associated Monitoring Data	Supported	Supported	Supported	Supported	Supported
Control Execution of a Workflow	Supported	Supported	Supported	Supported	Not Supported[10]

Table 2: Evaluation of state-of-the-art Workflow Projects

1. As stated in section 6.3.5, PMP is a script based workflow environment. Constructing workflows in PMP involves writing a PERL script, defining the operations of the workflow.

Hence PMP does not support the GUI based workflow specification mechanism neuGrid users require.

2. As stated in section 6.3.5, PMP is a script based workflow environment; hence modifying workflow entails the modification of the PERL script that defines the workflow.

3. PMP does support annotations in the code of the PERL script, but not visually as required by neuGrid users.

4. PMP does not have a library of modules. However users can select any algorithm they want to execute, because it is executed form the command line.

5. LONI Pipeline is a cluster-oriented tool, hence it does not have Grid planning and enactment functionality.

6. Kepler is a generic environment, which can be used for Grid execution. Kepler comes with built in support for Globus, but has been extended to other environments as well. Kepler can be integrated with Pegasus workflow planning toolkit, which provides data locality based scheduling (Pegasus is evaluated in section 6.4.2).

7. Taverna is a service-oriented workflow tool. neuGrid actors are executable files; hence a task based workflow tool is required. Service based workflow tool requires the actors of a workflow to be services themselves, hence in order to adapt such a solution tasks must be wrapped into services. Taverna support workflow planning for service based workflows by using Moteur (Moteur is evaluated in section 6.4.1).

8. Triana supports both task-based workflows and service-based workflows. However, the task based workflow enactment mechanism is based on GridLab GAT, which does not support the gLite middleware, which that is planned to be used in neuGrid.

9. PMP does not support Grid workflow planning and enactment and is a cluster based solution.

10. With PMP users execute pipelines from the command line, hence in order to stop the execution, users must terminate the command line session, and in order to restart users must manually stop and start the workflow.

6.3.7. Observations on Workflow Authoring Environments

In the light of the comparison of the workflow environments against neuGrid user requirements the following observations emerge.

6.3.7.1 LONI

LONI is a cluster level workflow execution tool. neuGrid Infrastructure is based on a Grid system, hence LONI Pipeline will not be able to enact authored pipeline on the neuGrid infrastructure (WP7). Hence LONI pipeline cannot be considered a part of the Pipeline Service environment as it is incompatible with the work in WP7. LONI Pipeline as documented in Appendix A, consists of two components: The user interface and the server module. The LONI Pipeline user interface fulfils most of the neuGrid user requirements, but the server module does not. Hence if the LONI Pipeline user interface can be decoupled from the server module, then LONI Pipeline interface can be used as part of the neuGrid Pipeline Service.

6.3.7.2 Taverna

Taverna is a service based workflow environment. The primary actors in neuGrid pipelines are task executables; hence some mechanism would be required to translate tasks into services in order to author pipelines in Taverna. neuGrid pipelines mandate a task based workflow environment, as per Task 5.3 of WP5. WP5 T5.3 aims at gridifying both the LORIS database as well as ensuring that the image analysis pipeline software works transparently over a Grid environment. Adoption of Taverna would create an incompatible solution with WP5.

6.3.7.3 Triana

Triana supports enactment of task based workflows, however there are some issues. Triana uses Gridlab GAT to enact pipelines; however a gLite adaptor for GridLab GAT does not exist. neuGrid infrastructure is based on gLite hence tools with support for gLite should be considered. The dependency of Triana on GridLab GAT is incompatible with one of the primary objectives of WP6, which is providing a Grid middleware agnostic and an independent solution.

6.3.7.4 PMP

PMP is a scripting based workflow environment. It does not fulfil numerous user requirements. It does not support Grid level enactment, it does not support visual construction of pipelines, and no validation is supported. All these requirements make it unsuitable for deployment in neuGrid.

6.3.7.5 Kepler

Kepler supports all user interface specific neuGrid user requirements. The backend of Kepler is designed to be generic. Kepler has been integrated with Pegasus workflow planner and supports enactment on Grid resources. Hence Kepler can provide an end-to-end task based workflow solution. Because all environments, except PMP support the required user interface requirements of neuGrid users, the final determinant factor is the support for Grid workflow planning and enactment. Hence the following sections explore this aspect and details related project in these domains.

6.4 Pipeline Gridification

The neuGrid infrastructure is Grid based. WP10 requires gridification of the neuGrid pipelines, hence planning for Grid workflow execution is an important aspect of the Pipeline Service. There are several technologies and approaches which can be considered, the following is a brief description of some of the technologies which are under consideration to grid-enable pipelines.

There is no standard definition of the term “grid-enabling”, but usually grid enabling of grid workflows is interpreted as the mapping of parallel parts of a pipeline onto grid resources, for optimal parallel execution, reduction of the pipeline size and execution latency by leveraging replicas of data produced and consumed and finally clustering and partitioning the pipeline for increased efficient usage of grid resources and speedy execution of the pipeline.

There are few direct user requirements for pipeline gridification, however there are strong requirements for pipeline gridificaiton in WP10, which deals with algorithms and pipeline gridification. There are some dependant technical requirements, which govern the functionality of the pipeline gridification mechanism, and are highlighted in section 6.4.3.

6.4.1. MOTEUR

Moteur [25] is an enactment engine for service-based SCUFL workflows. It provides the capability to parallelize web service workflows for Grids. It primarily supports gridification of service-based workflows by focusing on more parallel processing of SCUFL workflows.

6.4.1.1. Parallelization in Moteur

MOTEUR supports asynchronous calls to web services allowing it to proceed with the execution of a pipeline without waiting synchronously for web service invocations. MOTEUR implements the workflow parallelism approach on top of the latter. Workflow parallelism depends on the graph topology. For instance if we consider the simple example presented in Figure 18, processors P_2 and P_3 may be executed in parallel.

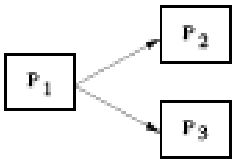

Figure 18: Simple Workflow

Another type of parallelism supported in Moteur is data parallelism. In a data-intensive application workflow, several actors may work on the same data sets. Moteur exploits this by executing all concerned components concurrently. The final type of parallelism support in Moteur is service parallelism. In this type of parallelism, if input data sets are completely independent for certain actors in a workflow, all the concerned actors are executed concurrently.

Moteur, since it is designed for web service based workflows, does not support scheduling of jobs against resources. It is implicitly assumed that workflow actors are wrapped by web services, which are pre-installed and deployed on specific resources. Moteurs' grid enabling mechanisms hence supports parallelism based on the workflow definition only, and in order to optimally use this, the users must deploy all actors manually. This increases the complexity of using the system for end users.

6.4.2 Pegasus

Pegasus [20] is a workflow planner for DAG based workflows. Unlike Moteur, which optimizes SCUFL workflows and adapts its parallelization mechanism towards web-services based workflows, Pegasus implements scheduling as well as graph topology parallelization for task based workflows. Pegasus introduces the concept of workflow compiling. Pegasus allows users to define workflows in a technology agnostic manner, called an abstract workflow. Pegasus compiles this abstract workflow into a concrete execution plan for a grid infrastructure by perusing the site catalogue of the Grid (which defines the sites in a Grid), the replica catalogue (which defines the physical location of datasets used in the workflow), and the transformation catalogue (which allows Pegasus to make decisions if some actors need to be moved to some sites prior to execution or the execution of a specific part of a workflow needs to be carried out in a specific site).

The workflow compiling process in other terms, includes finding the appropriate software and computational resources where the execution can take place, as well as finding copies of the data indicated in the workflow instance. The compiling process can also involve workflow restructuring geared towards optimizing the overall workflow performance as well as workflow transformation geared towards efficient data management. The result of the compiling process is an efficient executable workflow. The advantage of this approach for neuGrid is that the pipeline components are executable tasks, optimizing their scheduling against grid resources and enhancing parallelism in the pipeline will greatly reduce the processing time of the complete pipeline.

6.4.2.1. Grid Enabling in Pegasus

There are three primary techniques deployed in Pegasus to grid-enable a workflow. Pegasus supports pipeline reduction through data-reuse. If for some sequence of operations in a workflow, some data set is generated which already exists in the Grid environment, the entire sequence of operations is eliminated in the workflow. This feature is very useful in re-executing the workflow in response to some error or modifications. Parts of the intermediary output data are discovered through replica location services and whole sequences of operations, which produce the existing data, are eliminated. This decreases the number of jobs executed in a workflow. Other types of grid enabling in Pegasus include topological clustering of jobs workflow and logical partitioning of the workflow for increased concurrent execution across sites.

6.4.3. Matching Technical Requirements

There are 3 essential technical dependant requirements:

- Tool must support execution over Grid

There are numerous pipeline enactment solutions that are not designed for Grid applications hence any planner that is selected must support planning for Grid applications. This is a dependant requirement from WP7, which establishes a Grid infrastructure for the processing of the pipelines.

- Tool must support task based workflows

This is a dependant requirement from WP5. As NE executables consist of executable tasks; hence pipeline authored by the users will consist of tasks based workflows. Hence it is important for the workflow planner to be capable of planning for task-based workflows.

- Tool must be sufficiently adaptable in order to port it to other Grid infrastructures

This is a dependant requirement from WP6, which mandates grid agnostic tools. Grid agnostic solutions are preferable as the generic medical services will not be dependent and bound to specific Grid Middleware.

Technical Requirements	***Moteur***	***Pegasus***
Tool is Grid enabled	Yes	Yes
Tool support Task based workflows	No	Yes
Tool is Middleware agnostic	Yes[1]	No[2]

Table 3: Evaluation of Grid-Enabling Frameworks

1. Moteur is middleware agnostic as it enacts service oriented workflows. Services can be created in any environment, the middleware itself is irrelevant

2. Pegasus supports Globus and condor only. However the enactment information is stored in an open XML based format hence the information can be reused by client for workflow enactment on other middleware.

6.4.4 Suitability of Grid Enabling Frameworks

In the light of the comparison of the grid-enabling frameworks against the technical requirements for Grid enabling a pipeline, the following observations can be made.

Moteur is primarily service oriented SCUFL based workflow planner and enactment engine. As it is based on Taverna authored workflows, it is not suitable for neuGrid pipelines, without extensive SCUFL to task based workflow translation.

Pegasus is a task based workflow planner. However it is not middleware agnostic. It is primarily built for Globus and Condor. Hence in order to use Pegasus as part of the neuGrid Pipeline Service, it has to support gLite Grid middleware.

6.5 Enactment

Enactment is the final component of the Pipeline Service. Enactment deals with the actual execution of the user-defined workflows on Grid resources. Relevant projects are highlighted in this section.

6.5.1. Moteur

Moteur is an external enactment engine for SCUFL based workflows. It does not integrate into the Taverna environment rather, workflows have to be saved in Taverna and Moteur has to be invoked manually. Moteur parallelizes the workflow as discussed in section 6.4.1, and then enacts it accordingly. SCUFL based workflows are web-services based hence enactments constitute of exchanging SOAP messages with actor services in order to process the workflow. As previously stated, actor services are pre-installed and deployed on separate sites before a workflow can be created. This greatly increases the complexity of the system for neuGrid, for the following reasons:

1. neuGrid actors are executable processes, in order to use these processes in Moteur based environment, they first need to be wrapped in a web service.

2. These web services must be manually deployed on multiple sites for optimal distribution.

3. When new actor services are to be made available to users in order to support new applications, the first two steps must be repeated for each new executable process.

Due to manual and static deployment of actors, the infrastructure looses scheduling capability, which could have greatly enhance performance of workflows as well as more efficient use of the computing resources.

6.5.2 DAGMan

DAGman [18] is a workflow enactment engine for Condor. DAGman unlike Moteur does not do any parallelization or grid-enabling. However it is often used in conjunction with Pegasus, which does the required grid-enabling of the pipeline. DAGman works on pipelines, which are based on tasks rather than web-service actors, as is the case in Moteur. A directed acyclic graph (DAG) is used to represent the pipeline, which is a set of programs where the input, output, or execution of one or more programs is dependent on one or more other programs. The programs are nodes (vertices) in the graph, and the edges (arcs) identify the dependencies. DAGman is a workflow enactment engine specific to Condor and is included in the standard condor distribution.

6.5.3 GRIA

GRIA [34] is designed to facilitate the deployment of Grid application for businesses. GRIA has a Taverna plug-in making it an enactment engine for Taverna workflows. GRIA exposes two basic services: the Job Service and the Data Service. The Data service is designed to upload/download data from client machines, and the Job service publishes certain jobs, which are to be used by the users in pipelines. Using the published jobs it is possible to create workflows consisting of GRIA actors. Via the GRIA submission tool the workflows can be submitted and executed on Grid resources. For execution Torque/PBS and Condor are supported, however pipelines are not grid enabled, hence execution does not support Grid enabling of the workflows.

Primary limitations of GRIA are as follows:

1. GRIA does not include scheduling services. Rather a user has to develop a python script, which makes the scheduling decisions amongst sites. Eventually a single site is selected for workflow execution. Hence scheduling GRIA does not support fine-grained scheduling where certain components could execute in one site, and other components in others.

2. Additionally, GRIA is based on a non-GSI compliant PKI infrastructure. This means that all sites in Grid must run services developed for and deployed on the GRIA container.

3. Staging of processes is manual in GRIA. Since GRIA does not support fine-grained scheduling, the user has to manually stage executable actors to the site where the execution will take place.

6.6 Architectures

In the light of the technologies detailed in section 6.3, 6.4 and 6.5, we present tentative Pipeline Service architectures. The following architectures are designed to implement both the roles of the Pipeline Service, defined in section 6.1 and satisfy all essential user requirements, outlined in section 6.2.

After evaluation and review of existing related technologies the following four possible architectures are proposed. The advantages and disadvantages of each architecture are outlined.

6.6.1 GRIA Based Architecture

This Pipeline Service integrates Taverna, GRIA as well as Pegasus, as depicted in Figure 19. Taverna is used as the authoring environment as well as the environment where the user views and retrieves results. GRIA is used to enable the enactment and gridification of the Pipeline. The gridification is carried out by Pegasus. The Glueing Service (represented as the Execution Interface(SAGA)) is used to submit the actors for final execution.

Vanilla Taverna distribution supports only services based workflows, hence Taverna toolkit has the GRIA plug-in installed which allows the users to develop workflows using GRIA jobs which are exported via the Job Service. The GRIA Jobs can be individual task based actors. The architecture works as follows:

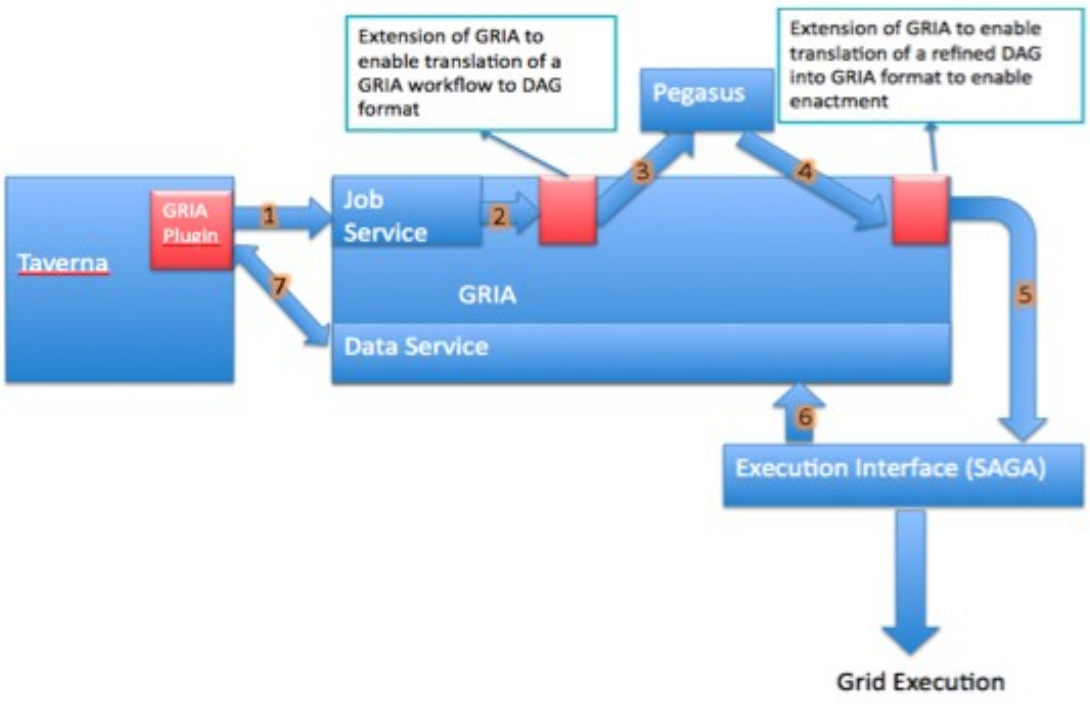

Figure 19: GRIA based Architecture

Taverna enables the creation of workflows using the GRIA actors and allows the initiation of the workflow execution (1). The execution of GRIA service from Taverna is managed via FreeFluo [22]. FreeFluo does web-service invocations against the Job Service of GRIA. These invocations on the Job Service are translated into a DAG via an extension to the GRIA Job Service (2). The DAG that represents the abstract pipeline, is passed to Pegasus (3). Pegasus returns a concrete execution plan, which is ready for execution on Grid resources. This plan is re-translated into GRIA format (4) which then can be enacted by GRIA on Grid resources via another extension which allows the submission of the concrete plan to a SAGA based execution interface (5). GRIA natively only support Torque/PBS and Condor. The final results of the execution can be downloaded via the GRIA Data service at the users Taverna instance (6,7).

The advantages of this architecture are that the development effort can concentrate on extending GRIA, by developing mechanisms for translation of GRIA workflows to DAG and vice versa. Additionally a mechanism needs to be devised to enable the execution of GRIA workflows against the Execution interface to be devised for this project. Moreover, GRIA also comes with a mature security infrastructure that can be implemented in the project.

The disadvantages include the introduction of overheads for translating GRIA workflows to DAG and vice versa. Additionally because GRIA is a container environment, and only two services are used the whole package may introduce overheads in the platform. GRIA, as mentioned in section 6.4.3, does not support fine-grained scheduling making it unsuitable for HPC Grid environments. GRIAs' security PKI infrastructure is incompatible with GSI, hence limiting the interoperability with other Grid middleware. GRIA has manual file staging to sites where the workflow execution will take place. GRIAs' Taverna plug-in is also developed for an old version of Taverna, which does not have drag and drop workflow creation making it less user friendly in creating workflows. The Pros and Cons of the approach are highlighted in Table 4.

Pros	Cons
Easy to use drag-&-drop environment	No Grid support
GRIA support for Condor, PBS	No native Pipeline Optimization, hence translation mechanism for Pegasus needs to be developed
Taverna integration with myExperiment	Overheads in translation
Mature PKI based security infrastructure	Data service provides stub for a provenance service, but it needs to be developed and customized. Hence support for a basic provenance is provided in the standard distribution

Table 4: Pros and Cons of the GRIA Based Architecture

6.6.2 Pegasus based Solution

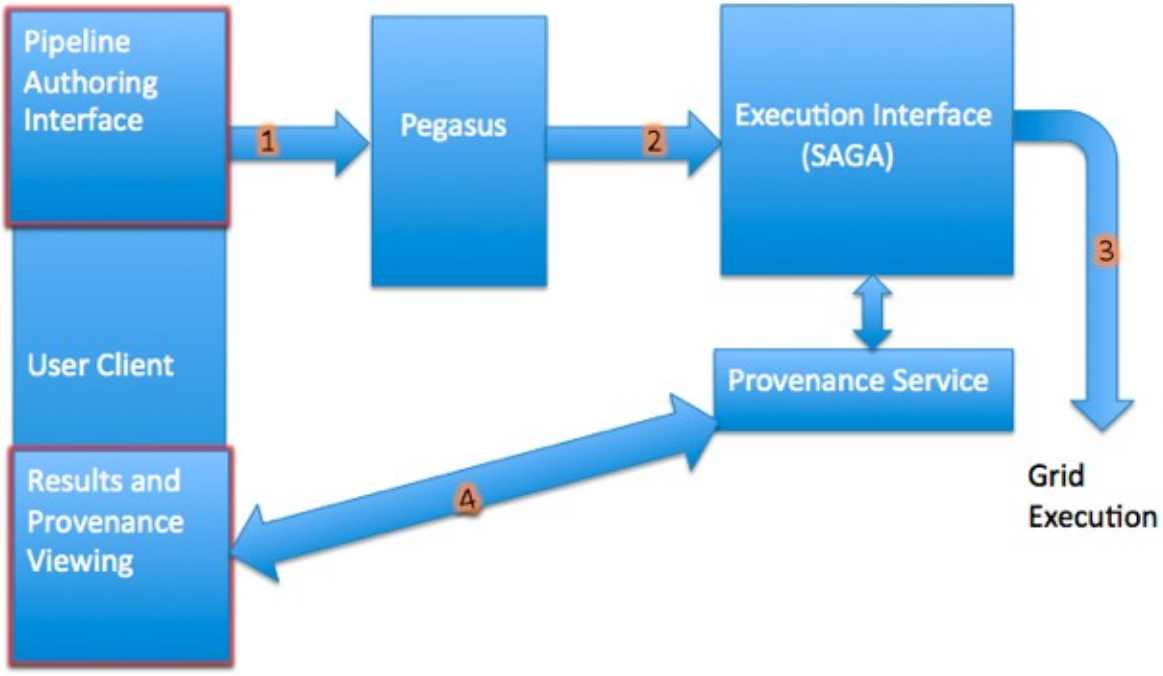

Figure 20: Pegasus based Architecture

Since Taverna enables the creation of web service oriented workflows, and NE tools constitute of

executable tasks another architecture can be proposed which focuses on providing a user-friendly environment, which allows the users to author and invoke workflows, this architecture is depicted in Figure 20. Kepler is a generic scientific workflow environment built without any leaning to the task based or service based workflow approach is suitable as the pipeline authoring interface. Current LORIS pipelines require a task-based approach to pipelines, however this should not limit the potential execution of service oriented workflows in the system. Hence selecting a generic environment is a suitable choice. The invoked workflows are natively stored in DAG format that can be directly passed to Pegasus without any need for translation (1). Pegasus grid-enables the abstract user defined pipeline and its output is used (2) to pass to the SAGA based execution interface that allows the eventual execution of the pipeline over a grid. Through the Provenance Service users would be able to look at the results and intermediary steps as well as outputs of the executed pipeline. The client interface will be rich enough to allow the re-execution of certain parts of the pipeline.

The advantages of this architecture are that the development effort can concentrate on providing a rich user interface rather than reinventing/extending existing middleware mechanisms. The interface can be designed form ground up to work with existing tools like Pegasus and have better integration with services which will be provided and developed in the project such as the Provenance Service, Knowledge service etc. The Pros and Cons of the architecture are highlighted in Table 5.

Pros	Cons
Easy to use Drag and Drop environment	Lack of Grid enactor, Pipeline Service enactor has to be coded
Straight forward integration with pipeline optimization	Pegasus is not a Grid middleware agnostic tool, however because of the Glueing Service this is abstracted.
Native use of task based workflow descriptions	
Pegasus integrates PASOA provenance service for historical workflow information	

Table 5: Pros and cons of the Pegasus based Solution

6.6.3 Taverna – Moteur - A-C Architecture

This architecture, as depicted in Figure 21, consists of using a complete end-to-end web-service based solution. Taverna is used as the environment for authoring workflows. The workflows are created against web service descriptions of the LORIS executables, which are exported by the A-C Web Service (1). In this architecture the user does not enact the pipeline from Taverna rather Moteur is used for grid enabling the web services based workflow before execution which is also done by Moteur (2). Because the A-C Web service exports the actor descriptions, enactments will be made against this web service, and this web service will translate invocations from Moteur into a DAG formatted concrete executable pipeline (3). This pipeline then can be submitted to the SAGA based execution interface. Because Moteur is an external enactment engine, users can not view results from the Taverna interface, this shortcoming would be complemented via another interface for viewing of the provenance data (4).

The advantages of this architecture include the reuse of Taverna and Moteur, which are already well integrated for pipeline authoring and grid enabling. However in order to author a pipeline web service descriptions need to be provided for the LORIS executables which can be an extensive effort. Additionally, the web service wrappers need to be manually deployed and distributed. This severely limits scheduling capability and leads to inefficient use of computing resources. Additionally provenance tracking is more difficult in this architecture. Because the process execution provenance is provided by Moteur while the outcome of the processes is provided by SAGA, both these information needs to be merged and provided to the users. The means to provide provenance information to users will call for another interface because Moteur being an external enactment engine does not show provenance information in the Taverna interface like FreeFluo does. The pros and cons are highlighted in Table 6.

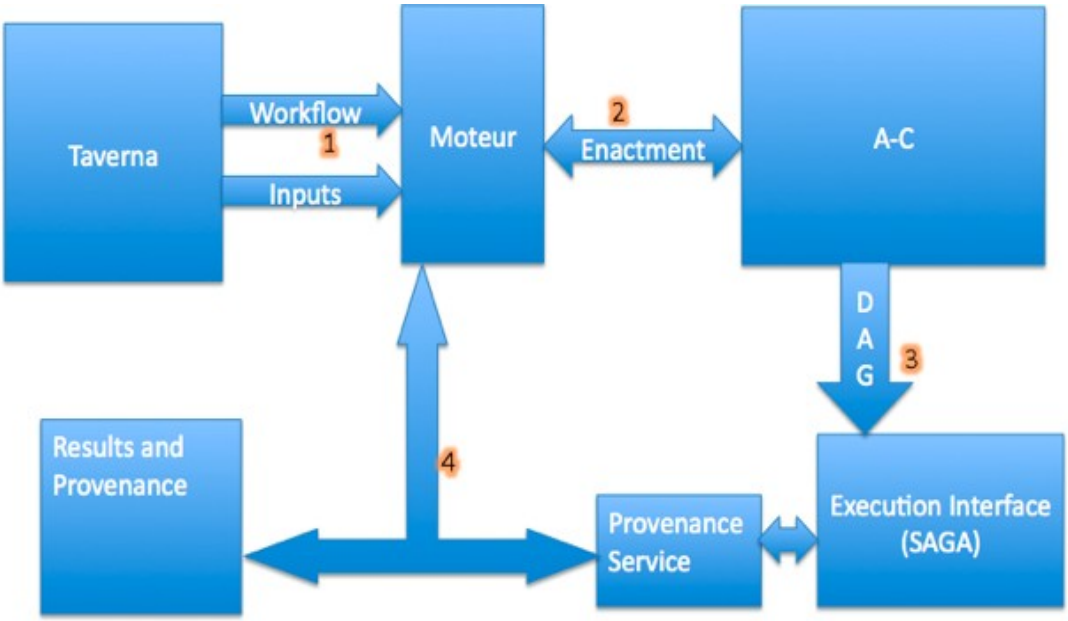

Figure 21: Taverna-MOTEUR based Architecture

Pros	Cons
Mature drag and drop environment	Overheads in translation
Good integration with pipeline optimization (Moteur)	External enactor looses visual enactment from Taverna
	No staging of actors or data sets

Table 6: Pros and cons of the Taverna-Moteur-A-C Architecture

6.6.4 Triana based architecture

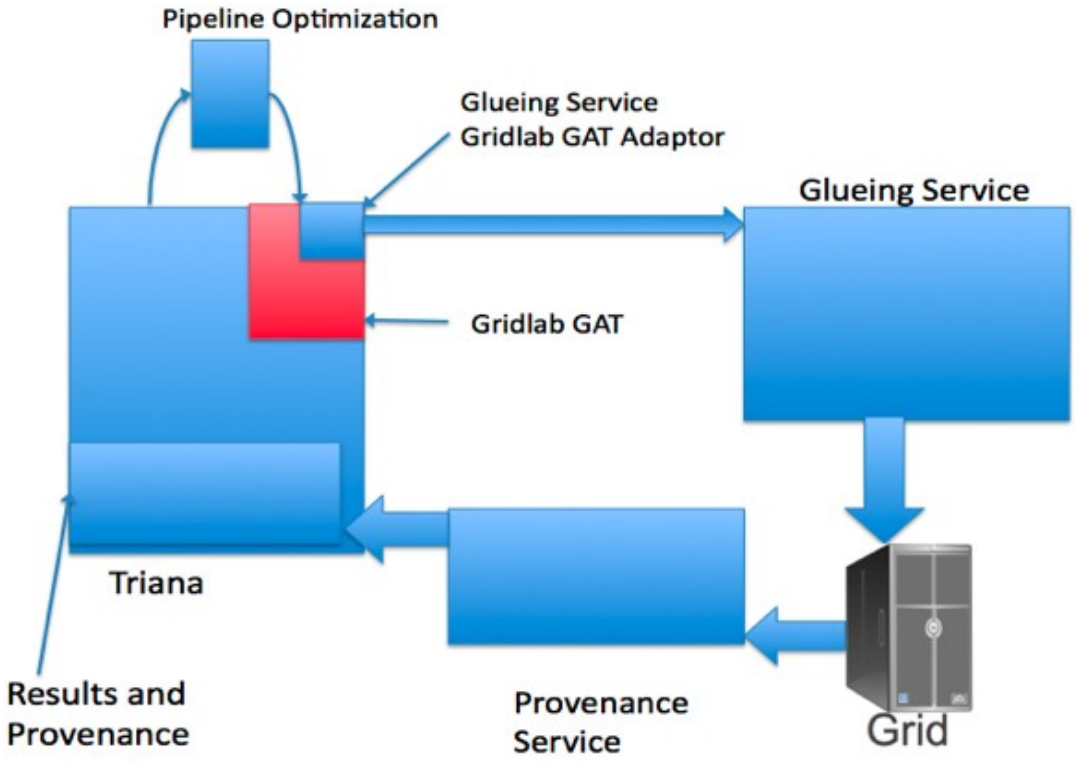

Figure 22: Triana based Architecture

In this architecture, as depicted in Figure 22, the user authors pipelines in Triana. Once the user invokes the enactment of a pipeline, the pipeline is first grid-enabled and optimized via Pegasus. Then the enactment is carried out via the Gridlab GAT adaptor for the Glueing service. The Glueing service eventually executes the tasks on the Grid.

Pros	Cons
Mature drag and drop environment	Overheads in translation
Support for task based workflows	Glueing service Grid lab GAT adaptor needs to be developed
	Overheads due to two levels of Grid API's Grid lab GAT which is integrated in Triana, and SAGA which's frontend is provided by the glueing service

Table 7: Pros and cons of the Triana based architecture

This proposed architecture uses Triana as the pipeline authoring environment. Since task based

workflows can be invoked via only Gridlab GAT, hence a Gridlab GAT adaptor for the Glueing service needs to be developed. Apart from this translation mechanism needs to be developed where the pipeline is translated from the Triana format to Pegasus format and then translated back. These translation mechanisms and integration with Pegasus needs to be developed. The pros and cons of the architecture are highlighted in Table 7.

6.7 Recommendations

In the light of the user requirements and the pros and cons of the architectures, the second proposed architecture that makes use of Pegasus, a mature workflow planner and Kepler as a generic workflow-authoring environment seems to fulfil all major neuGrid requirements. The principle factors in selecting the second architectures include:

1. Use of a generic and mature scientific workflow authoring environment
2. Use of a mature task based grid enabling toolkit
3. Extent of integration between both outlined technologies (Kepler and Pegasus)
4. Integration of Pegasus with PASOA [28] (Provenance Tracking)
5. There is a clear separation of concerns in this architecture; each component can be changed without affecting any other component as the project evolves. Kepler is the current authoring environment however this environment can be replaced with any other suitable workflow authoring environment.

On the other hand principle factors which discourage adaption of Architecture 1 and 3 include:

1. Lack of scheduling capability in both GRIA and Moteur
2. Focus on static actor workflows in Moteur (actors are pre-installed in sites)
3. Lack of interoperability of GRIA with other middleware, due to incompatible security infrastructures
4. Complex provenance gathering in Architecture 3
5. Manual file staging in Architecture 1, and none in Architecture 3

Hence Architecture 2 seems to provide a suitable service for neuGrid. The various components under consideration are: Kepler as the workflow authoring environment, PASOA as the mechanism for provenance gathering and Pegasus to grid-enable the workflow. The limitations of the mentioned software are outlined below which need to be addressed during the course of the project.

6.7.1 Kepler

Kepler provides a generic environment to create pipelines/workflows. The work on integrating Kepler and Pegasus was done. In the paper it is mentioned that the director was a work in progress and the director is not shipped with the default distribution of Pegasus. It is unclear as to what is missing in the director right now, as the director becomes available limitations can be studied and solutions explored. Additionally Kepler will be ported to a suitable format, for integration into a portal for neuGrid. Kepler will be the primary workflow authoring environment while, the Portal will integrate information from the several neuGrid services such as provenance service, knowledge service, metadata service and workflow status information, as well as visualization for results.

However this architecture is not bound to Kepler alone. An advantage of this architecture is that there is a separation of concerns. The Grid planning and enactment is handled by separate independent components. Moreover, the planning and enactment component takes workflow in a specific format; hence the user interface can be flexible because in any case transformations have to be made, as shown in Figure 23.

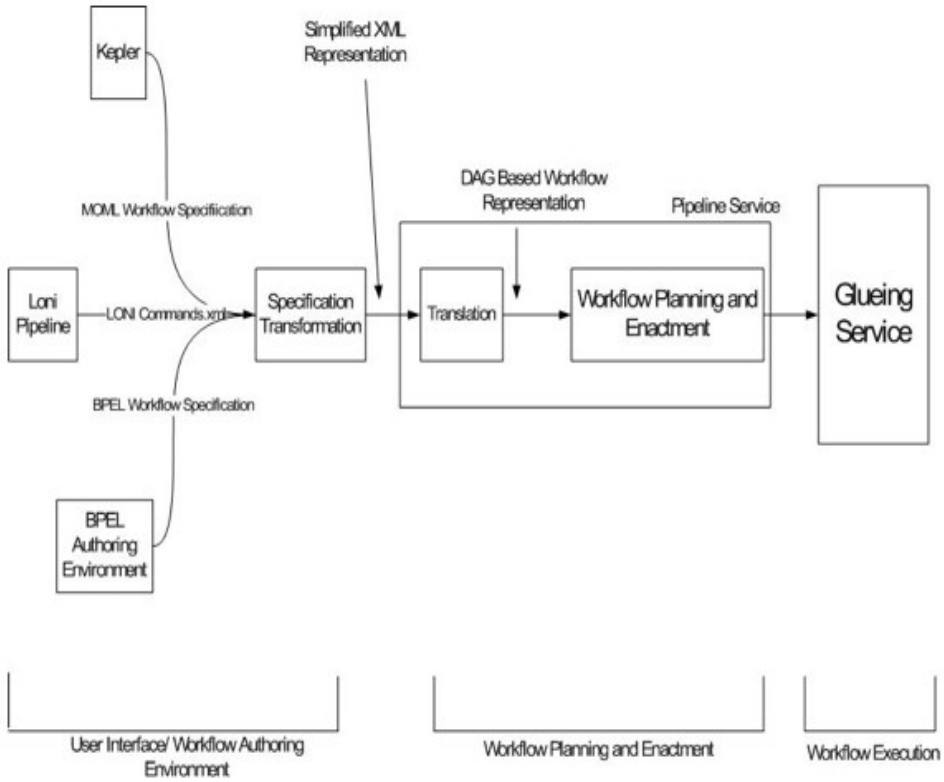

Figure 23: Flexible Pipeline Service Architecture

6.7.2 Pegasus

Pegasus provides a mature framework to grid-enable abstract user defined pipelines to Grids. However Pegasus has support for Condor DAGMan only, and can talk to Globus via the Condor-G extension. The NeuGrid Pipeline Service should be middleware agnostic, hence requires that Pegasus should be extended to support other middleware as well. Because in the neuGrid architecture middleware will be shielded via the gluing service, Pegasus needs to be ported to work with the glueing service.

6.8. Pipeline Service Design

There are three primary components in the architecture outlined in section 6.6.2: The authoring environment, Pegasus and the grid submission/enactment mechanism.

As stated in section 6.7.1, the authoring environment is flexible. The documented design, presented in this section, features two authoring environments, Kepler and LONI. Due to the closed source nature of LONI Pipeline and the open source nature of Kepler, the Pipeline Service is completely integrated with Kepler. The Pipeline Service is used as an external service for LONI Pipeline.

The components of the Pipeline Service are outlined in Figure 24. The interaction starts with the authoring of a pipeline, which the user wants to execute on the Grid (1). Authoring can be done in numerous tools, the prototype implementation presented in this section, uses Kepler and LONI Pipeline as the authoring environment. As previously pointed out in Figure 23, the architecture is flexible and any suitable authoring environment can be accommodated.

After authoring the pipeline, the user invokes submission of the pipeline (2). In this case, several things happen: (3) first the authored pipeline, which is represented in a Modelling Markup Language (MoML) format (in case of Kepler) or in LONI Pipeline XML (in case of LONI) is transformed into a simple XML based workflow format, which is passed to the Pipeline Service. The Pipeline Service translates the specification into a workflow object, via an API, which will be provided as part of the Pipeline Service (detailed in section 6.8.2.2). The workflow object is translated into a DAX file, via the Pegasus DAX API. Pegasus is used as a workflow-planning tool in this environment. This DAX file represents the abstract workflow the user defined. Pegasus using the Grid site catalogueue, the transformation catalogueue and the grid replica catalogueue, plans the workflow into a concrete executable workflow. The following operations are carried out by Pegasus on the workflow.

1. Tasks are mapped to individual grid sites, depending on availability of task actors and/or study set replicas or partial workflow outputs.
2. Portions of the workflow are mapped to specific grid resources, depending on the computing platforms and computing resources provided by the sites.
3. Enhances workflow specification by including data staging actors to stage data between sites.
4. Enhances workflow specification by including provenance actors for provenance collection.

The Pipeline Service uses this information and updates the workflow specification and enacts the workflow via the Pipeline Service Enactor. Figure 25 sums up the transformations that happen to a workflow until its executed. This diagram will be references in the following section where a complete explanation of the architecture is detailed.

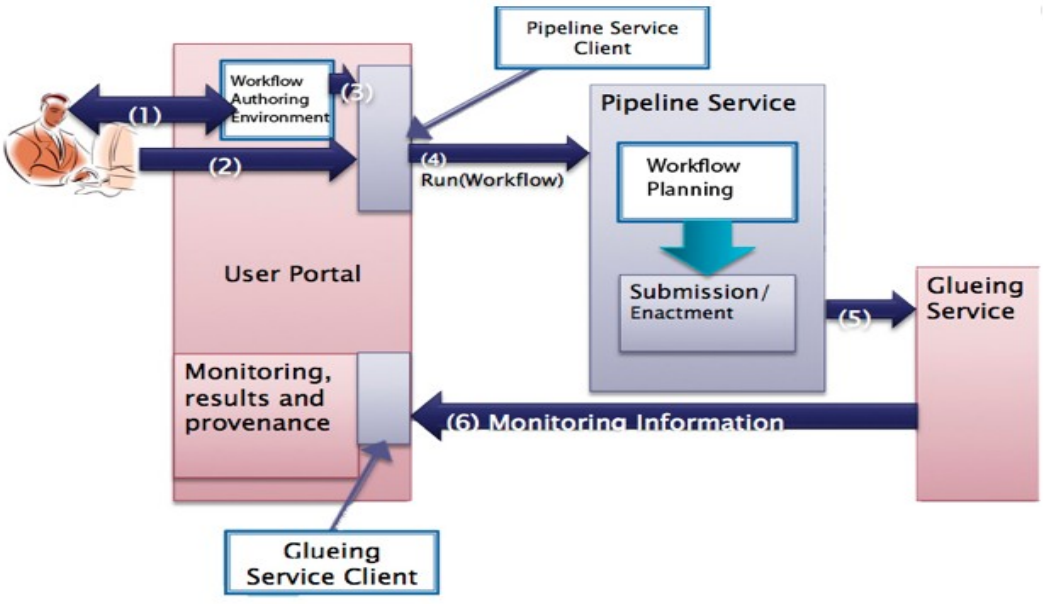

Figure 24: Pipeline Service

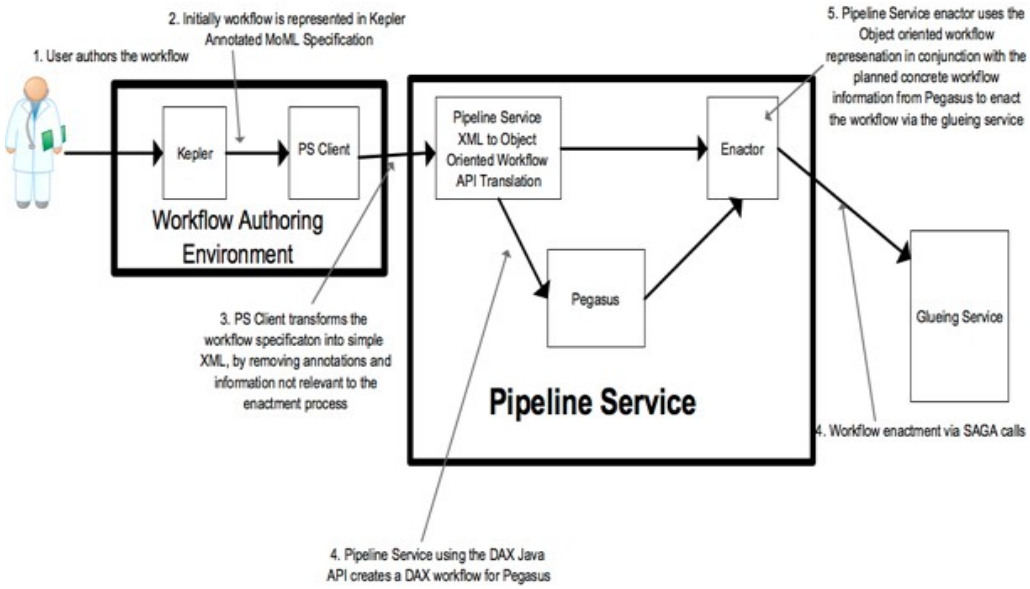

Figure 25: Transformations in Pipeline Authoring to Enactment (using Kepler)

6.8.1 End to End architecture description

The architecture is explained via a sample workflow that is authored by the user. The focus of this section is to illustrate how a workflow would be executed from an abstract description provided by the user to the final output of execution results. Two authoring environments are used in this example, LONI and Kepler.

6.8.1.1 Workflow Authoring Environment

Users author workflows in Kepler and LONI, in a graphical drag and drop environment. A screenshot of the Kepler environment is shown in Figure 26. Kepler will provide numerous actors and algorithms, which the users want to use in their workflows. Algorithms must be described in XML format to Kepler in order for them to be accessible in the interface.

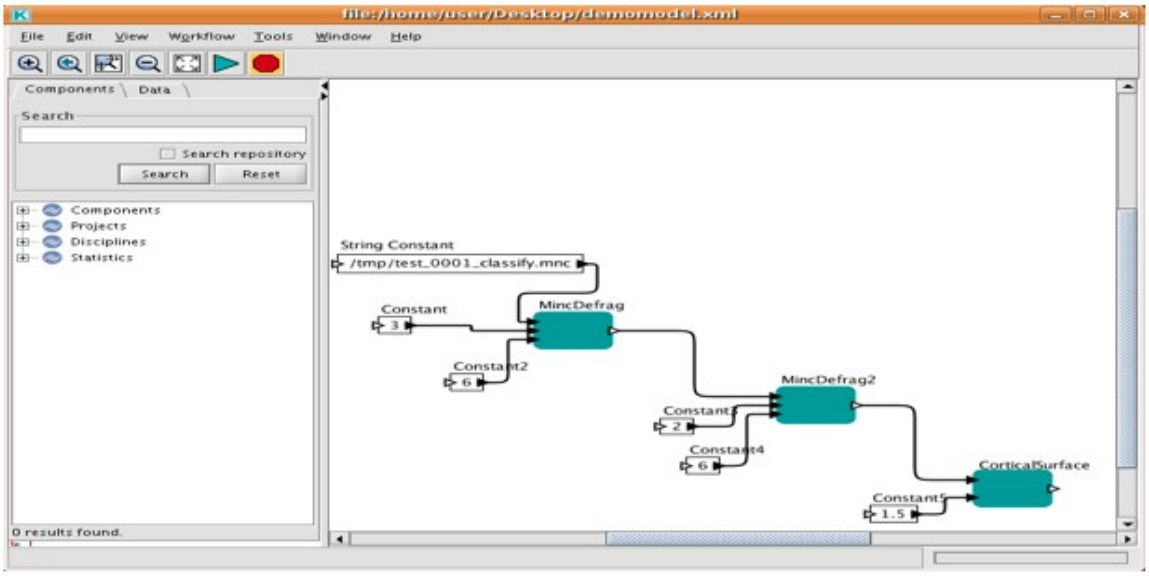

Figure 26: Kepler Interface

A sample description of the mincdefrag algorithm is shown in the following listing.

Salient features of the algorithm specification include, the entity name, which will define an abstract entity in the environment. The input "port" types, which are input parameters to the algorithms and the output port, which is the output parameter of the algorithm. If port types mismatch, Kepler does not allow the connection of the actors. XML based descriptions will be included for all algorithms so that users can create workflows with ease. Additionally, the interface will be enhanced to allow for dynamic data set selection. This will be detailed in the Querying service specification.

```
<?xml version="1.0"?>
<entity name="MincDefrag" class="ptolemy.kernel.ComponentEntity">
    <property name="entityId"  value="urn:lsid:kepler-
project.org:actor:545:1"
     class="org.kepler.moml.NamedObjId"/>
    <property name="documentation"
class="org.kepler.moml.DocumentationAttribute">
    </property>
    <property name="class"
value="eu.neugrid.pipelineservice.mincdefrag"
     class="ptolemy.kernel.util.StringAttribute">
    <property name="id" value="urn:lsid:kepler-project.org:class:545:1"
     class="ptolemy.kernel.util.StringAttribute"/>
    </property>

    <property name="ClassifyFile"
class="org.kepler.moml.PortAttribute">
      <property name="direction" value="input"
class="ptolemy.kernel.util.StringAttribute"/>
      <property name="dataType" value="unkown"
class="ptolemy.kernel.util.StringAttribute"/>
      <property name="isMultiport" value="false"
class="ptolemy.kernel.util.StringAttribute"/>
    </property>
```

Listing 1

There are three primary areas in the Kepler interface. All three areas are highlighted in the Figure 27. The area marked by a red rectangle is the authoring workspace. In this area user will drag and drop algorithms, define parameters and inputs and finally define the order of execution. The green marked area is the actor/algorithm search toolbox. From this area user can search and select algorithm which he wants to use in the workflow. The orange marked area, represents workflow control options. The green "play" button starts the execution of the workflow. This involves translating the Kepler workflow into a series of transformations before it can be executed.

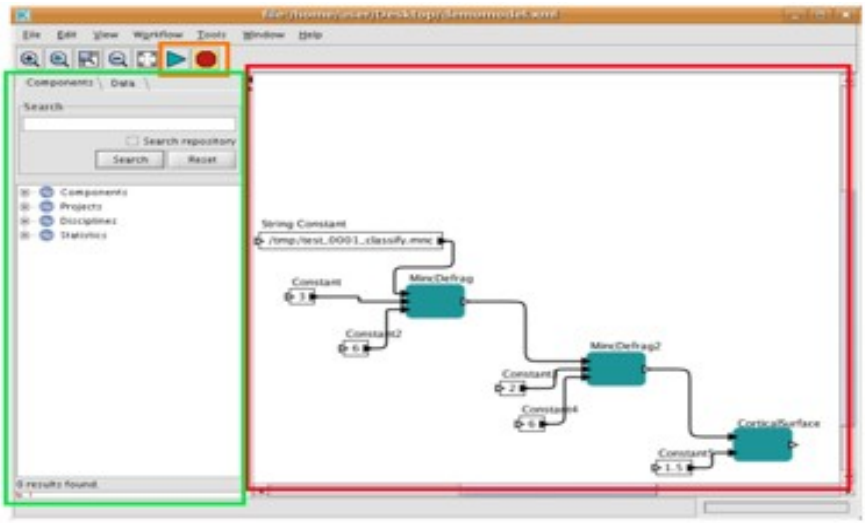

Figure 27: Marked Kepler Interface

Kepler uses the MoML specification to represent workflows. When the workflow is enacted it is first saved in a MOML format. Part of the MoML format of this workflow is shown in listing 2.

```
<link port="MincDefrag.ClassifyFile" relation="relation"/>
   <link port="MincDefrag.value" relation="relation2"/>
   <link port="MincDefrag.value2" relation="relation3"/>
   <link port="MincDefrag.output" relation="relation4"/>
   <link port="MincDefrag2.ClassifyFile" relation="relation4"/>
   <link port="MincDefrag2.value" relation="relation5"/>
   <link port="MincDefrag2.value2" relation="relation6"/>
   <link port="MincDefrag2.output" relation="relation7"/>
   <link port="CorticalSurface.ClassifyFile" relation="relation7"/>
   <link port="CorticalSurface.value" relation="relation8"/>
   <link port="String Constant.output" relation="relation"/>
   <link port="Constant.output" relation="relation2"/>
   <link port="Constant2.output" relation="relation3"/>
   <link port="Constant3.output" relation="relation5"/>
   <link port="Constant4.output" relation="relation6"/>
   <link port="Constant5.output" relation="relation8"/>
</entity>
```

Listing 2

The full workflow is shown in Appendix B.

The same workflow can be created in LONI Pipeline interface, and the same workflow is shown in Figure 28.

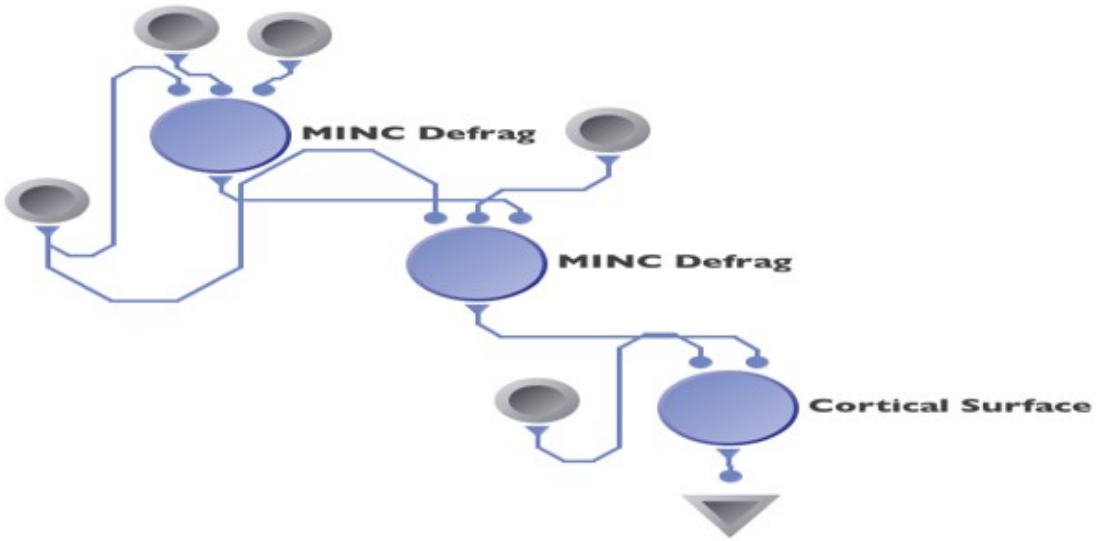

Figure 28: LONI Pipeline authored workflow

This LONI Pipeline authored workflow is stored in a XML file, an excerpt of the file is shown in Appendix C. The LONI Pipeline Interface is detail in Appendix A.

6.8.1.2 Submission to the Pipeline Service Client

The Pipeline Service client is a component integrated with the authoring environment. Because of the open source nature of Kepler, the Pipeline Service is integrated into Kepler which enables it to support seamless execution of a workflow over the Pipeline Service. Once a workflow is submitted from Kepler, the Pipeline Service client, transforms the Kepler MoML workflow into a simplified XML workflow and submits it to the Pipeline Service. The XML format the Pipeline

Service uses is the pure MoML specification, which does not contain Kepler specific annotations. The non-annotated MoML specification was selected due to its simplicity. The specification includes a basic xml construct for actors, input values, output values and relations and link amongst the actors. The complete simplified workflow is shown in Listing 3.

The Pipeline Service client after translating the Kepler annotated MoML workflow into simple non-annotated MoML invokes the Pipeline Service and submits the workflow for enactment.

For LONI Pipeline, because of its closed source nature, an external Pipeline Service client is used to transform the LONI Pipeline XML specification into the required simplified XML format used by the Pipeline Service.

```
<?xml version="1.0" encoding="UTF-8"?>
<workflow>
<actor name="MincDefrag" class="eu.neugrid.pipelineservice.mincdefrag"/>
<actor name="MincDefrag2" class="eu.neugrid.pipelineservice.mincdefrag"/>
<actor name="CorticalSurface" class="eu.neugrid.pipelineservice.cortical_surface"/>
<actor name="String Constant" class="ptolemy.actor.lib.StringConst"/>
<actor name="Constant" class="ptolemy.actor.lib.Const"/>
<actor name="Constant2" class="ptolemy.actor.lib.Const"/>
<actor name="Constant3" class="ptolemy.actor.lib.Const"/>
<actor name="Constant4" class="ptolemy.actor.lib.Const"/>
<actor name="Constant5" class="ptolemy.actor.lib.Const"/>
<ConstantValue value="/tmp/test_0001_classify.mnc"/>
<ConstantValue value="3"/> <ConstantValue value="6"/>
<ConstantValue value="2"/>
<ConstantValue value="6"/>
<ConstantValue value="1.5"/>
<relation name="relation"/>
<relation name="relation2"/>
<relation name="relation3"/>
<relation name="relation4"/>
<relation name="relation5"/>
<relation name="relation6"/>
<relation name="relation7"/>
<relation name="relation8"/>
<link port="MincDefrag.ClassifyFile" relation="relation"/>
<link port="MincDefrag.value" relation="relation2"/>
<link port="MincDefrag.value2" relation="relation3"/>
<link port="MincDefrag.output" relation="relation4"/>
<link port="MincDefrag2.ClassifyFile" relation="relation4"/>
<link port="MincDefrag2.value" relation="relation5"/>
<link port="MincDefrag2.value2" relation="relation6"/>
<link port="MincDefrag2.output" relation="relation7"/>
<link port="CorticalSurface.ClassifyFile" relation="relation7"/>
<link port="CorticalSurface.value" relation="relation8"/>
<link port="String Constant.output" relation="relation"/>
<link port="Constant.output" relation="relation2"/>
<link port="Constant2.output" relation="relation3"/>
<link port="Constant3.output" relation="relation5"/>
<link port="Constant4.output" relation="relation6"/>
<link port="Constant5.output" relation="relation8"/>
</workflow>
```

Listing 3

6.8.2 Pipeline Service

The main role of the Pipeline Service is to provide seamless enactment of user defined workflows over a Grid. The Pipeline Service provides its functionality via a web service based interface. The interface of the Pipeline Service is described in section 6.8.2.1. Once a client submits a workflow in the simplified MoML format, the Pipeline Service converts the XML-based specification into a workflow object. The object-oriented workflow API is described in section 6.8.2.2. In 6.8.2.3 we detail the interaction of the Pipeline Service with Pegasus and how the planned workflow is eventually enacted. The enactment approach is detailed in 8.2.4.

6.8.2.1 Pipeline Service Web Service Interface

The following documents the methods provided by the Pipeline Service.

Non-interactive execution

String run(Workflow userWorkflow, int sessionID);
String terminate(String workflowID, int sessionID);
Int registerSession();

Interactive Execution

String enact(Task usertask, int sessionID);

registerSession Method

This method is provided to maintain state at the Pipeline Service side. The method returns a unique integer number, which identifies the session for a particular client.

run method

The run method does several things: It first invokes Pegasus to grid-enable the workflow defined by the users. The grid-enabled pipeline is then enacted via the Glueing service. Due to the lack of appropriate meta-scheduling adaptors the enactments have to be made by the Pipeline Service. The *run* method returns a unique string, which identifies the specific workflow. In the context of a session, multiple workflows can be executed.

terminate method

The terminate method, as the name suggests terminates the execution of a workflows in a specific session.

enact Method

This method is provided for interactive execution of workflows. The previous functions do not provide for interactive executions, because user cannot gain feedback from the execution until the entire workflow has been enacted by the Pipeline Service. However when the user wants to test a new pipeline, and wants to study the behaviour it may be useful for the user to enact it form the authoring environment and study behaviour actor by actor.

6.8.2.2 Workflow API

```
WorkflowAPI()Vector Workflow;
   int AddTask(Task newTask, int PredecessorID)
   void addPredecessor(int TaskID, int PredecessorTaskID)
void addPredecessor(int TaskID, Vector PredecessorTaskID)
    Task getTask(int TaskID)
    void setTask(Task newTask)({)void removeTask(int id)
void removePredecessor(int taskID, int PredecessorID)
}Vector Workflow;
   int AddTask(Task newTask, int PredecessorID)
   void addPredecessor(int TaskID, int PredecessorTaskID)
void addPredecessor(int TaskID, Vector PredecessorTaskID)
    Task getTask(int TaskID)
    void setTask(Task newTask)({)void removeTask(int id)
void removePredecessor(int taskID, int PredecessorID)
}void removeTask(int id)
void removePredecessor(int taskID, int PredecessorID)
}Vector Workflow;
   int AddTask(Task newTask, int PredecessorID)
   void addPredecessor(int TaskID, int PredecessorTaskID)
void addPredecessor(int TaskID, Vector PredecessorTaskID)
    Task getTask(int TaskID)
    void setTask(Task newTask)({)void removeTask(int id)
void removePredecessor(int taskID, int PredecessorID)
}void removeTask(int id)
void removePredecessor(int taskID, int PredecessorID)
}void removeTask(int id)
void removePredecessor(int taskID, int PredecessorID)
}
```

Listing 4

WorkflowAPI Class

The workflowAPI class will be used to describe workflows specified by the user in the workflow authoring environment. An explanation of the proposed methods is provided. To demonstrate the use of the workflow API and the Task class a sample workflow is considered. Graph structure of the workflow is shown in Figure 29.

The workflow starts with an actor, named *actor1*, which processes the input to the workflow *workflow.in*, the output of the workflow is fed into two further actors. Both actors process the output of actor1 simultaneously. The output of the workflow consists of two files *actor2.out* and *actor3.out*.

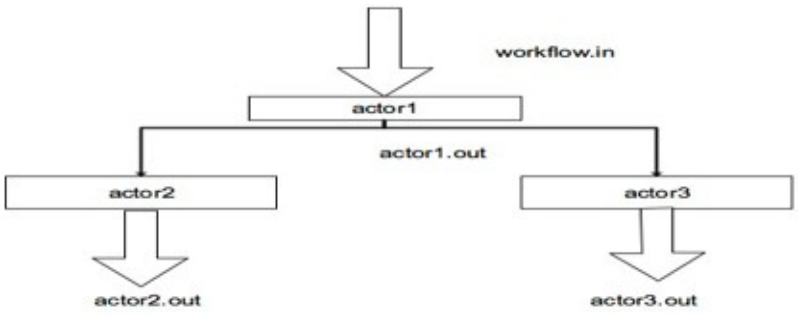

Figure 29: Workflow example

addTask

With the addTask method the user would be able to add tasks to a workflow.

```
1.  workflow = new Workflow();
2.  actor1 = new Task();
3.  actor2 = new Task();
4.  actor3 = newTask();
5.  actor1.setExecutable(`/shared/actor1‘);
6.  actor1.setInput(`/shared/workflow.in‘);
7.  actor1ID = actor1.getID();
8.  actor1.setOutput(‘actor1.out’);
9.  actor1.setArgumets(`-in‘);
10. actor1.setlogs(`actor1.errlog‘,’actor1.outlog‘,’actor1.exelog‘);
11. workflow.addTask(actor1,null);
```

Listing 5

In this the shown code listing numerous thing happens, the line 1-4, define the actors and the workflows. In Line 5-10 *actor1* is initialized with the executable, input, outputs, and any arguments that need to be passed to the executable are set. The location of the logs are defined as well. In line10 the task is finally added to the Workflow object named *workflow*.

AddPredecessor

There are two means of specifying a dependency; one is through the constructor or explicitly through this function. In the workflow shown in figure 29, both *actor2* and *actor3* are dependent on the output data from actor1. Listing 6 demonstrates how the scenario can be expressed.

Listing 6 shows the usage of both the constructor method of adding a predecessor task and using the AddPredecessor method. Line 1 to 11 shows the initialization of *actor2* and *actor3*. Line 12 shows how a predecessor is specified via the addTask method. Line 13-14 show how AddPredecessor method is used.

```
1.  actor2.setExecutable(`/shared/actor2‘);
2.  actor2.setInput(`/shared/actor1.out);
3.  actor2.setOutput(‘actor2.out’);
4.  actor2.setArgumets(`-in‘);
5.  actor2.setlogs(‘actor2.errlog‘,’actor2.outlog‘,’actor2.exelog‘);
6.  actor3.setExecutable(`/shared/actor3‘);
7.  actor3.setInput(`/shared/actor1.out‘);
8.  actor3.setOutput(‘actor13out’);
9.  actor3.setArgumets(`-in‘);
10. actor3.setlogs(‘actor3.errlog‘,’actor3.outlog‘,’actor3.exelog‘);
11. actor3ID = actor3.getID();
12. workflow.addTask(actor2, actor1ID);
13. workflow.addTask(actor3, null);
14. workflow.AddPredecessor(actor3ID, actor1ID);
```

Listing 6

Overloaded AddPredecessor method

The AddPredecessor function used in the code listing, takes the task ID of the task and a single predecessor. This is suitable for our workflow, however for workflows where there are more than 1 predecessors like shown in Figure 30.

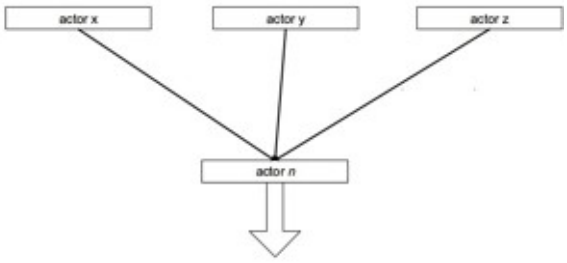

Figure 30: Advanced Predecessor Example

In order to cater for these scenarios an overloaded method has been provided where an array of predecessor tasks can be specified.

getTask, and setTask methods

After a workflow has been defined, it is possible to amend some tasks, like specifying a new input file to a certain task. In this scenario, the user has to maintain a taskID. A sample use of the workflow class in the scenario is documented in listing.

```
1. Task t1 = workflow.getTask(actorID);
2. t1.setInput('/shared/newinput');
3. workflow.setTask(t1);
```

Listing 7

The Workflow class will maintain an internal directed acyclic graph data structure, where vertices would tasks. The setTask function will replace the t1 class with the appropriate equivalent class the DAG data structure.

removeTask method

The removeTask method removes a task from the workflow. However this may lead to workflow consistency issues because the predecessor tasks and the successor tasks will not be linked. In this case the developer has to use the getTask, setTask functions to modify properties of the predecessor and successor tasks accordingly.

removePredecessor method

The *removePredecessor* method allows the user to remove a predecessor of a workflow, may be used with the removeTask function in order to cleanup legacy defined dependencies.

In the architecture shown in Figure 24 the Pipeline Service will use the workflow API to convert the simplified MoML into a programmatic DAG. Before we proceed with the enactment in step 5 in figure 24, the Pipeline Service uses Pegasus to plan the workflow.

6.8.2.3 Pegasus

Pegasus supports numerous planning techniques for Grid workflow. In a Pegasus environment, a user submits an application-level description of the desired workflow in an abstract format. Pegasus uses various data from the Grid including the site catalogue, transformation catalogue and the replica catalogue. The site catalogue contains information about the sites in a Grid. The

transformation catalogue contains abstract to physical mappings for operators in a workflow and finally the replica catalogue is used locate the copies of the operators and data used in the workflow.

The following steps are used by Pegasus to create a concrete workflow for execution

1. Firstly, Pegasus consults the site catalogue (SC) to find which Grid nodes are available in the Grid. Site catalogue can be either a Grid information service or a text file containing descriptions of the sites available in the Grid.

2. Workflows are executed multiple times on a Grid. It is possible that a subset of the data that will be generated in the current workflow has been generated in a previous execution. Hence Pegasus queries the RLS service (replica catalogue) to find instances of data products, which will be generated in the workflow. If such instances are found, the tasks, which lead up to the generation of the data, are eliminated.

3. The previous step assumes that it is more efficient to access the data then to re-compute it. Given the workflow, a site selection is performed. This selection can be done based on the available resources and their characteristics as well as the location of the required input data. Site selection is based on a few standard algorithms which the users can choose, the algorithms include: random, round-robin and min-min selection. These algorithms can be applied to the selection of the execution site as well as the selection of the data replicas. The selection algorithms make use of information available in site catalogue, the transformation catalogue and replica catalogue.

4. Pegasus provides an option to cluster jobs together in cases where a number of small granularity jobs are destined for the same computational resource. During clustering we consider only independent tasks, so that they can be viewed by the remote execution system as a single entity. These tasks also need to be destined for the same execution system [26].

5. The abstract workflow contained only nodes representing computations. Since the workflow can be executed across multiple platforms and since data need to be staged in and out of the computations, Pegasus augments the workflow with tasks that explicitly perform data transfers.

After all the previous optimization, the final submit file is generated. This file can be directly submitted to DAGman, or through Condor-G to the Grid.
The focus of neuGrid is to support enactment of workflows in a middleware agnostic manner. The Pegasus mapper is used in the Pipeline Service. The Pegasus mapper does all the planning and the enactment is done via the Pipeline Service enactment engine.

The same workflow is transformed for Pegasus into a DAX. Pegasus provides a Java API for creating DAX workflows programmatically. In order translate the Workflow object the Pipeline Service creates into a format Pegasus can understand, the Pegasus DAX API is used to dynamically create workflows. The following is an example of the same workflow being created in the DAX API.

```
String id1="ID0000001";

        //create a job
        Job job=new Job (NAMESPACE,MINCDefrag,VERSION,id1);
        //add the arguments to the job
        job.addArgument(new Filename(FA));
        job.addArgument(new PseudoText("3"));
        job.addArgument(new PseudoText("6"));
        job.addUses(new Filename(FA,LFN.INPUT));
        Filename f=new Filename(FAo,LFN.OUTPUT);
        f.setRegister( false );
        job.addUses(f);
       //add the job to the dax
        dax.addJob(job);

        //create second  job
        String id2="ID0000002";
        Job job=new Job (NAMESPACE,MINCDefrag,VERSION,id1);
        //add the arguments to the job
        job.addArgument(new Filename(FAo));
        job.addArgument(new PseudoText("2"));
        job.addArgument(new PseudoText("6"));
        job.addUses(new Filename(FAo,LFN.INPUT));
        job.addUses(new Filename(FC,LFN.OUTPUT));
        //add the job to the dax
        dax.addJob(job);

        //create third job
        String id3="ID0000003";
        job=new Job (NAMESPACE,CorticalSurface,VERSION,id3);
        //add the arguments to the job
        job.addArgument(new Filename(FC));
        job.addArgument(new PseudoText("1.5"));
        //add the files used by the job
        job.addUses(new Filename(FC,LFN.INPUT));
        job.addUses(new Filename(CorticalOut,LFN.OUTPUT));
        //add the job to the dax
        dax.addJob(job);

      //add the relationships between the jobs
        dax.addChild(id2,id1);
        dax.addChild(id3,id2);
```

Listing 8

Listing 9 shows the DAX file that is generated, which is passed to Pegasus to plan the execution. The final DAG workflow that is generated contains concrete execution information. A DAG workflow contains multiple submit files for individual actors. Pegasus inserts staging information as well. Listing 10 is a specific submit file for a single actor.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- generated: 2008-09-29T08:13:51+01:00 -->
<!-- generated by: root [??] -->
<adag xmlns="http://pegasus.isi.edu/schema/DAX"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://pegasus.isi.edu/schema/DAX
http://pegasus.isi.edu/schema/dax-2.1.xsd" version="2.1" count="1"
index="0" name="daxfile" jobCount="3" fileCount="0" childCount="2">
<!-- part 1: list of all referenced files (may be empty) -->
<!-- part 2: definition of all jobs (at least one) -->
  <job id="ID0000001" namespace="neugrid" name="mincdefrag"
version="0">
    <argument><filename file="test_pid_classify.mnc"/>3 6</argument>
    <uses file="test_pid_classify.mnc" link="input" register="true"
transfer="true" type="data"/>
    <uses file="output" link="output" register="false" transfer="true"
type="data"/>
  </job>
  <job id="ID0000002" namespace="neugrid" name="mincdefrag"
version="0">
    <argument><filename file="output"/>2 6</argument>
    <uses file="output" link="input" register="true" transfer="true"
type="data"/>
    <uses file="min2defrag" link="output" register="true" transfer="true"
type="data"/>
  </job>
  <job id="ID0000003" namespace="neugrid" name="corticalsurface"
version="0">
    <argument><filename file="min2defrag"/>1.5</argument>
    <uses file="min2defrag" link="input" register="true" transfer="true"
type="data"/>
    <uses file="corticalsurface" link="output" register="true" transfer="true"
type="data"/>
  </job>
<!-- part 3: list of control-flow dependencies (may be empty) -->
  <child ref="ID0000002">
    <parent ref="ID0000001"/>
  </child>
  <child ref="ID0000003">
    <parent ref="ID0000002"/>
  </child>
</adag>
```

Listing 9

As we can see, in Listing 10, Pegasus has replaced abstract actor information with concrete paths pointing to specific Grid executables, as well as concrete input files and output files. The site on which the executable will execute is given in the “Pegasus_site”.

```
##########################################################
##################
# PEGASUS WMS GENERATED SUBMIT FILE
# DAG : daxfile, Index = 0, Count = 1
# SUBMIT FILE NAME : new_rc_tx_mincdefrag_ID0000001_0.sub
##########################################################
##################
environment =
GLOBUS_LOCATION=/vdt/globus;LD_LIBRARY_PATH=/vdt/globus/lib;
#arguments =  -P 4 -p 1 base-uri se-mount-point
arguments = " -P 4 -p 1 base-uri se-mount-point"
copy_to_spool = false
error =
/home/irfan/dags/irfan/pegasus/daxfile/run0004/new_rc_tx_mincdefrag_ID0
000001_0.err
executable = /opt/pegasus//bin/linux/transfer
input =
/home/irfan/dags/irfan/pegasus/daxfile/run0004/new_rc_tx_mincdefrag_ID0
000001_0.in
log = /tmp/daxfile-08937.log
notification = NEVER
output =
/home/irfan/dags/irfan/pegasus/daxfile/run0004/new_rc_tx_mincdefrag_ID0
000001_0.out
periodic_release = (NumSystemHolds <= 3)
periodic_remove = (NumSystemHolds > 3)
priority = 5
submit_event_user_notes = pool:local
transfer_executable = false
universe = scheduler
+pegasus_generator = "Pegasus"
+pegasus_version = "2.1.0"
+pegasus_wf_name = "daxfile-0"
+pegasus_wf_time = "20080929T081715+0100"
+pegasus_wf_xformation = "pegasus::transfer"
+pegasus_job_class = 3
+pegasus_job_id = "mincdefrag_ID0000001"
+pegasus_site = "local"
queue
##########################################################
##################
# END OF SUBMIT FILE
##########################################################
############  #####
```

8.2.4 Pipeline Enactment

The planned workflow is finally enacted, and the following is an excerpt form the enactment engine. The enactment engine uses Glueing service SAGA compliant calls to enact the workflow.

```
 String exe = tcC.getTransform(task.executable);
String gm  = new String("gridsam");
CreateJob cj = new CreateJob();
cj.setAdaptor("gridsam");
cj.setApp(exe);
cj.setArg(args);
CreateJobResponse cjr = stub.createJob(cj);
String jobid = cjr.getCreateJobReturn();
Run rn = new Run();
 rn.setJobId(jobid);
RunResponse rsn = stub.run(rn);
System.out.println(rsn.getRunReturn());
Monitor mn = new Monitor();
mn.setJobId(jobid);
 MonitorResponse mrsn = stub.monitor(mn);
String loc = mrsn.getMonitorReturn();
```

Listing 11

6.9 Research Issues

6.9.1 Introduction

e-Science workflows are broadly characterized as complex workflows which are both data and compute intensive. E-Science workflows are complex because they require a large number of processes and transformations and have a large number of data dependencies amongst them. neuGrid workflows, such as the CIVET (Cortical Thickness Pipeline) are e-Science workflows. neuGrid workflows can be further categorized as data mining workflows, because they carry out a sequence of transformations on raw image data to retrieve valuable and significant information.

The enactment and execution of e-Science workflows, such as those supported in neuGrid, on Grid resources faces many challenges. One of the challenges is inefficient resource usage and long application turnaround times when executing these workflows. Potential research issues and potential solutions are highlighted in this section. The focus in this section is on addressing the scalability of neuGrid or other data mining workflows in Grids.

This section discusses potential research problems and solutions covering:

1. Scalability of the neuGrid workflows.

2. Exploring the potential for integrating machine learning based methods in the planning of the workflows to produce more efficient workflows, reducing time and increasing efficiency.

3. neuGrid workflows are authored by non-technical Grid users. Users author pipelines according to application requirements. Authoring Grid workflows with just application requirements may lead to less adaptive and inefficient Grid workflows. To build more efficient and adaptive Grid workflows, numerous design considerations need to be taken into account, which significantly increases the complexity of authoring Grid workflows. Hence the transformation and adaption of the original specification into a concrete Grid executable plan is a research issue.

Figure 31 shows the outline of a typical Grid data mining application. Figure 31 can be used to express the cortical thickness pipeline in neuGrid, which is used to identify neurological diseases in patients. This workflow will be the primary application, which will be used to benchmark the approaches and mechanism developed during the project.

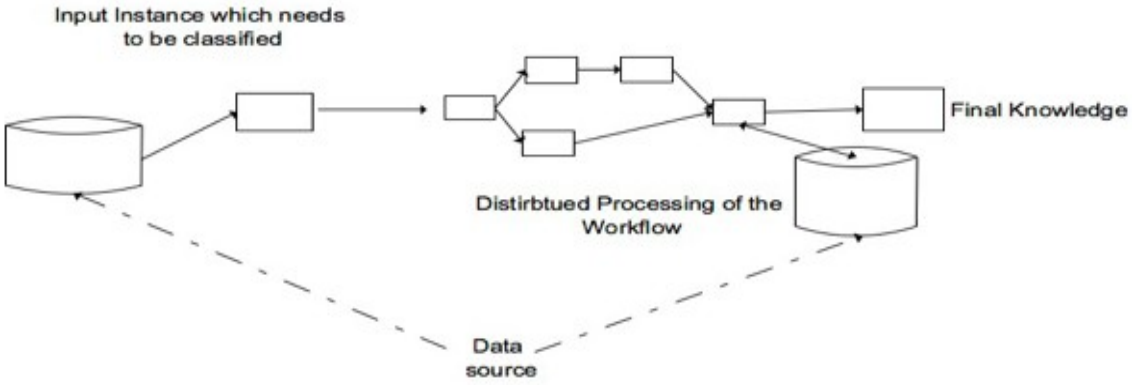

Figure 31: Outline of a Grid Data Mining Application

The application proceeds as follows: a patient needs to be diagnosed; the appropriate brain scans are retrieved and processed over a series of computations called a pipeline or a workflow to retrieve certain parameters which include parameters such as left hemispheric native thickness, right hemispheric native thickness and the mid-surface with cortical thickness asymmetry map. These parameters are classified against a control set of images with certain values. In data mining terms, the images in the control set represent certain points in a multi-dimensional space. The task of the classification algorithm, which is performed manually by statisticians, is to find the closest point from the unknown instance to the control set points. With near proximity to a certain control set point the nature of the instance can be identified.

It is planned in neuGrid to enable scientists to run the same process for thousands of images, ideally against a larger control set to allow for more accurate diagnosis. This can create scalability problems both in terms of computational and storage capabilities. Scaling up Grid data mining applications is a complex research issue. Many tradeoffs need to be considered and many approaches can be explored and adapted. For example in the context of the neuro-imaging pipeline, the instance which needs to be classified can be reduced dimensionally through feature selection, reducing both the multi-dimensional distance calculation and processing time, however feature selection may impact the accuracy of the results. Alternatively highly granular workflows can be constructed which cater for more efficient parallelization at the cost of increased queue waiting times. A further possibility is to develop more advanced schedulers that focus on near optimal scheduling for data intensive processes in the pipeline at the cost of compute intensive processes or vice versa. Yet another possibility is to explore efficient replica management.

Many approaches can be adapted and explored, however only certain aspects, specifically those which can be applied to a broader set of Grid applications will be explored in this research (detailed in section 6.9.2). The software that will be used to carry out the experiments is a Grid Pipeline Service, which is detailed earlier in this document. The Pipeline Service detailed in this document uses state-of-the-art workflow planners, which use only static information about the Grid resources, workflow actors and data locations, as shown in Figure 32. This produces a workflow, which is not inherently scalable as only static Grid information is used. Creating more intelligent workflow planners will be explored, as illustrated in Figure 33. The intelligent planner would use numerous data sources besides the static Grid information state of the art planners use. Historical workflow execution data, as well as application or workflow specific data will be combined into aggregate search space on which the search for an optimal workflow will be carried out.

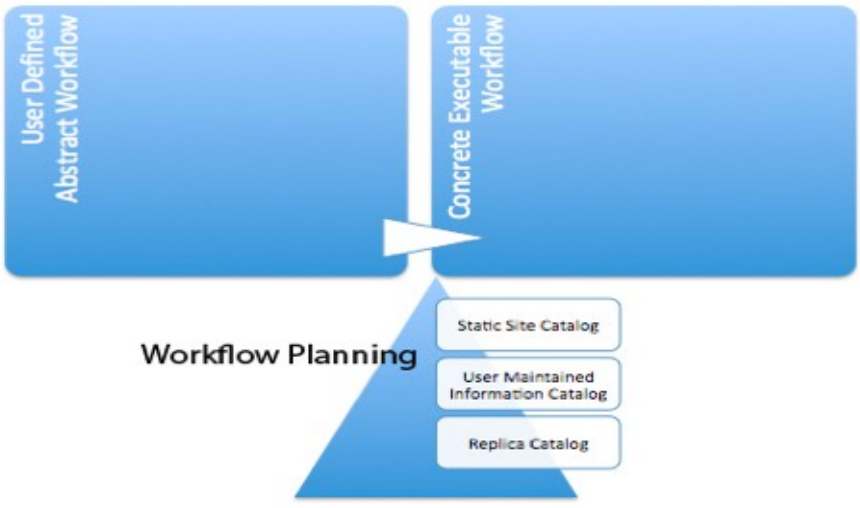

Figure 32: Workflow Transformation in Sate of the Art Workflow Planners

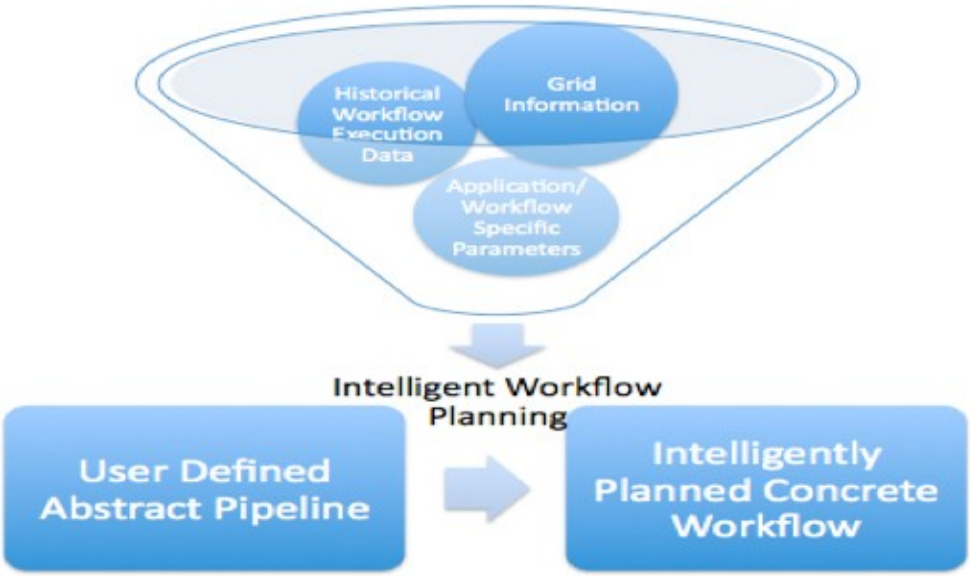

Figure 33: Intelligent Workflow Planning

6.9.2 Towards Intelligent Workflow Planning

The state of the art workflow planners, as outlined in section 6.4, try to improve upon the static concurrent execution of workflows, by trying to inculcate information about the Grid environment, like the available sites, replicas of data, clustering of granular jobs etc. But the information used by these tools is still static. More dynamic and intelligent workflow planning has been envisioned [16]. In [23] the workflow generation problem has been cast as an AI planning one in which the goals are the desired workflow outputs and the operators are the application components. An AI planning system is initialized with a representation of its current environment, a desired goal state, and a repository of operations it can take to achieve the goal state.

The planning system searches for a valid, partially ordered set of operations that will transform the current state into one that satisfies the goal. In terms of workflow planning, the operations are the actors in the workflow, and the goal state is the final result that is computed. The goal of effective workflow planning is the optimum configuration of actors against constraints that are enforced by dynamic Grid resources and preconditions, which are specified by application dependencies. Finding the optimum workflow description in this dynamic decision space in Grids is a multi-dimensional search problem as there are a myriad of parameters that can affect the execution of a workflow. Machine learning approaches such as genetic algorithms or ensemble learning are effective approaches for negotiating multi-dimensional search spaces, as they offer parallelized searching of a search space. Hence this research will explore the applicability of these methods in scaling grid data mining workflows.

It is envisioned that intelligent workflow planning techniques can provide high-quality solutions, partly because they can search several solutions and return the best ones found, and because they use heuristics that will likely guide the search to good solutions. However in contrast existing grid data mining applications frameworks include only rudimentary workflow planning if at all. The authors in [23] identified the following key areas, which needed to be addressed in future Grid workflow management including more efficient knowledge capture thereby enhancing usability in enabling more abstract definitions of workflows and improving robustness in terms of enhancing adaptiveness of the workflows in the dynamic Grid. Access rights in multi-organizational Grids was highlighted as well and

finally the problem of workflow scalability was identified. The authors [23] stated that primary issues with workflow scalability are both the large amounts of data they deal with and the scale of the workflows themselves which contribute to the problem's complexity. In a Grid environment often a pool of workflows is executing which also creates complex scheduling problem when workflows are not optimized.

Of the five categories highlighted by the authors, we focus on the scalability and robustness aspects of Grid workflows. In [21] the authors reiterate similar issues. They state that workflow performance has two aspects: efficiency and robustness. Efficiency deals with the ability to quickly bind a task to a grid resource and robustness deals with the ability to handle exceptions in a workflow without failing completely. One major bottleneck in workflow performance, they state, is the issue of data transfer, not just input data sources but also how efficiently data is moved between the tasks in a workflow. This is severely impacted by the workflow design [24][19]. Fine-grained workflows generate large volumes of data transfers amongst tasks, whereas coarse-grained workflows may result in inefficient resource usage.

In [24] it is stated that advances in scalability in workflow execution are required and advances will have to occur in multiple dimensions. The identified dimensions include: efficiently describing large scale workflows, scaling the number of resources involved in a workflow execution and finally efficiently handling increasing number of workflow actors.
In the light of this discussion, future work on the Pipeline Service will focus on creating more robust and flexible approaches, inspired from machine learning, to create more flexible workflows. As a sample study, we will explore scaling the cortical thickness pipeline/workflow from the neuGrid project. Future focus of the work will be to scale workflow pools as envisioned by researchers.

6.9.3 Suitable Machine learning Approaches

Numerous machine learning approaches can be explored to refine workflow specifications. However in order to leverage machine learning effectively, metrics need to be defined, which will determine the search space the machine learning algorithm will operate in. These metrics will represent certain characteristics of workflows.

6.9.3.1 Metrics and Fitness Function

Often for measuring the scalability of a data intensive workflow, the data throughput of a workflow is measured. Lower data throughput means a workflow keeps a large amount of data locally while it is processing it. This means that the total amount a workflow segment can process is proportional to the storage capability of the site where the workflow is executing. This leads to unscalable workflows as the scalability of such a workflow is directly proportional to the storage capability of the site. For scaling such workflow cleanup jobs in workflows have been introduced to increase scalability [31] . Other data related metrics include inter-site transfers, large inter-site transfers mean high communication latency of processes leading to slower execution. Intra-site transfers are significant as well. Although intra-site communication latencies are smaller than inter-site latencies however large number of intra-site communication means processes spend a large time in suspended state waiting for data.

Compute related metrics are important as well. As users generate workflows from an application point of view, not all processes in the workflow are equally compute intensive; a common solution for this problem is to cluster granular jobs together [26]. However state of the art workflow planners use static information about Grids to plan workflows and cluster jobs. A machine learning approach will use execution feedback to determine the best possible clustering strategy. Job clustering is also important in increasing or decreasing granularity of a workflow. Increased granularity means that more Grid resources will be able to be utilized during execution however at the same time, increase granularity means more scheduling delays. This is a typical trade-off situation, which may differentiate between scalable workflow execution and inefficient execution. Such complex search problems are ideal for

machine learning approaches.

Thus there are two primary classes of metrics, which will be used to guide a machine learning approach: data based and processing based. However numerous other metrics can be considered as well, esp. application specific metrics may be included when generating a new iteration of a workflow. The fitness function used to evaluate new workflows will be a combination of various metrics.

6.9.3.2 Suitable Machine Learning Approaches

Workflow optimization is an incremental process, as has been stated before it is also a multi-dimensional search problem. Machine learning approach such as evolutionary approaches, ensemble learning and composite machine learning algorithms are of significance to this problem. Genetic algorithms generate populations of candidate solutions, evaluate fitness and improve the next generation by selecting high-value parents, and crossing over and mutating them to generate a new pair of children. A workflow maybe represented as a direct acyclic graph. Mutations and crossover operations may be carried out via graph manipulations. A generation will be a set of workflows with characteristics but the same workflow task sequence. Fitness will be quantified by executing a workflow.

Fitness function itself can be dynamic. Since this is a multi-dimensional search problem. It is possible that certain parameters in the fitness function have more influence on the performance as other parameters. Hence in order to identify such features, feature selection on the fitness function itself can be of value. Feature construction that is creating a new feature from existing feature may be of value as well. For instance a single metric for data efficiency can be constructed dynamically from disparate data centric measurements. Such methods have been widely used in numerous domains [32].

Ensemble learning methods are based on statistics and are primarily used for classification. Ensemble methods, like Genetic algorithms, are suitable in traversing complex multi-dimensional search spaces and use statistical measurements to produce iterations of a candidate solution. Popular methods in Ensemble learning include bagging and boosting. In bagging a classifier is generated from different subsets of a data set, randomly drawn, and the efficiency of the classification process is defined by taking the accuracy of the entire ensemble. Similar methods can be applied to workflows too. Different population of workflows can be generated and co-evolved however using different fitness functions for subsets of the population; this may lead to a highly parallel approach to discovering suitable fitness functions which may be used in future to refine workflows.

As has been discussed, incremental, generational machine learning approaches such as genetic algorithms and ensemble learning are suitable for the workflow optimization problem. However in order to guide these approaches toward acceptable solutions correct fitness functions are required. Fitness function will consist of numerous metrics, which directly impact the performance of a workflow. However machine learning approach will be applied to discovery acceptable fitness functions. Methods such as case-based reasoning, feature selection and construction as well as genetic programming may be suitable. Numerous machine learning algorithms and composite machine learning algorithms can be applied, however we will explore and shortlist which of the approaches are most suitable. Those will be deployed and tested in prototype systems.

6.9.3 Methodology

The experimental infrastructure used will consist of prototype algorithms encoded into the Pipeline Service and a physical computing infrastructure that will be setup for the project studied. The cluster middleware Condor [17] will be used to create multiple pools of clusters. The Pipeline Service will apply the encoded mechanisms to the workflow being executed and the physical infrastructure will be used to execute the workflow and appropriate measurement for the quantitative analysis will be taken.

The methodology is illustrated in Figure 34.

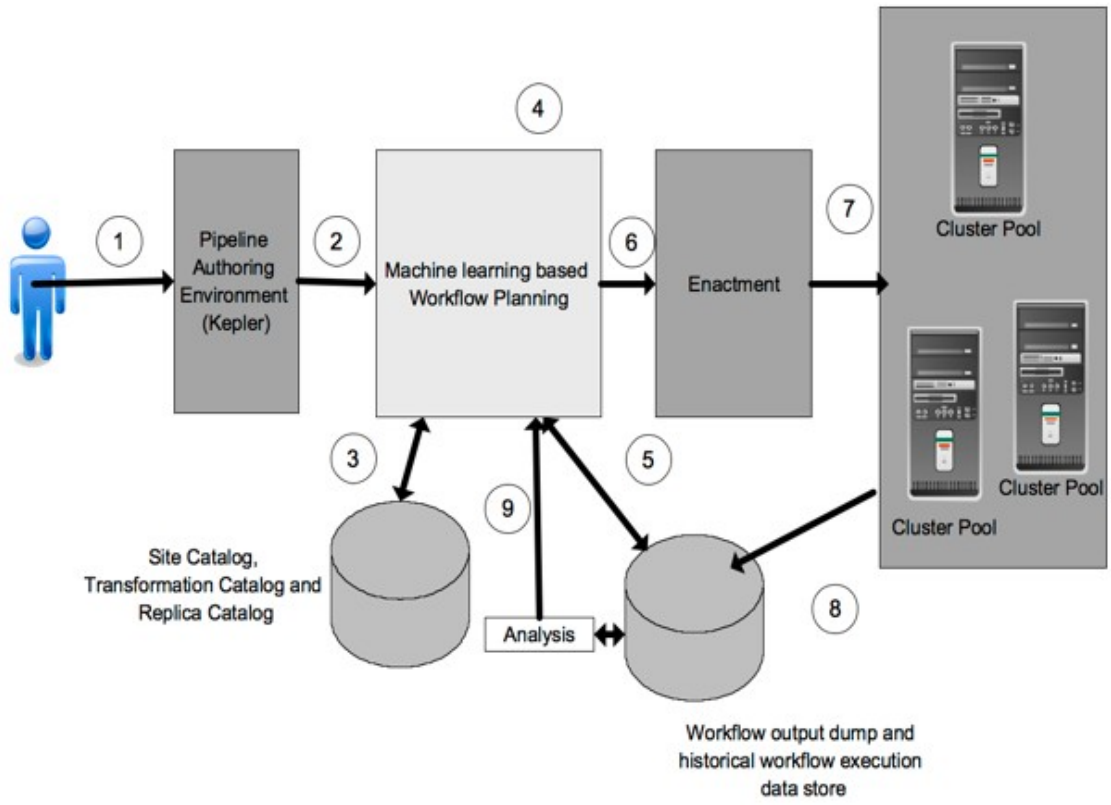

Figure 34: Research Methodology

At (1), the user defines the workflow in the pipeline-authoring environment; these workflows will be primarily neuGrid specific neuro-imaging workflows. After the user defines the workflow, he submits it for execution. Because user defined workflows are not directly executable in Grids hence it is first transformed at (2) by Pegasus into a concrete executable form. Pegasus uses various Grid specific data sources, shown in (3) such as the site catalogue, transformation catalogue and the replica catalogue to plan the workflow concretely, Pegasus is detail in section 6.2.2.6. In a traditional Grid workflow framework the workflow would be enacted in this form. However because the focus of this thesis is to explore scaling of grid data mining workflows using machine learning approaches, the Pegasus workflow planner would be replaced by an intelligent machine learning based workflow planner which would formulate a concrete execution plan based on not only static Grid information, but also relevant dynamic Grid information, historical workflow execution and application specific data. The machine learning approach will try to find optimal workflow according to a user-defined fitness function. Fitness functions will consist of number of parameters. Certain parameters which characterize a workflow need to minimized such as queue wait times, inter-task data transfers and some others need to be maximized such as task granularity, parallelism etc. A discussion on possible fitness functions is presented in 2.4.1.

After the machine learning approach is applied many iterations of the workflow can be generated and enacted on Grid resources. Execution logs and output data will be retrieved and stored in (6). Enactment, as shown in (7) in the project will be done against a Grid of Condor clusters. Condor is a popular cluster middleware, and numerous pools will be deployed to create a Grid.

Execution specific data is dumped to a data store (9). The data store will be used to analyze the efficiency of the workflow. This data will also be used for analysis of the approach, and improvements will be made to the implemented approach in response to deficiencies in the algorithm.

6.10 Conclusion

The Pipeline Service is an important component of the WP6 Generic Medical Services. The role of the Pipeline Service is to enable authoring of neuro-imaging pipelines, gridify and plan the enactment of the pipeline, enact the pipeline and finally allow users to retrieve and view results.

The design of the service is guided by the relevant neuGrid user requirements. The design is also compliant with the WP6 Services Design Philosophy. Related state-of-the-art projects were reviewed and evaluated. The architecture that most suited the user and technical requirements was selected. The selected architecture has numerous features, which make it suitable for neuGrid. The architecture promotes a separation of interests, where the authoring environment is completely decoupled from the gridification and enactment engine. Numerous authoring environment can be integrated which include LONI Pipeline, Kepler or a web-based authoring interface. At the other end, the design integrates seamlessly with other WP6 services, including the Glueing and Provenance Service.

Future research issues have also been identified in this document. These issues will be explored during the development of the Pipeline Service. The focus of the research would be to make the neuGrid Pipeline Service more scalable and efficient than compared to the existing state-of-the-art related projects.

6.11 References

[13] W3C XML Protocol, http://www.w3.org/2000/xp/

[14] Allen, G., D. Angulo, T. Goodale, T. Kielmann, and A. Merzky. Gridlab: Enabling Applications on the Grid. *Lecture Notes in Computer Science, 2002*

[15] Altintas, I, C Berkley, E Jaeger, and M Jones. Kepler: An Extensible System for Design and Execution of Scientific Workflows. *Scientific and Statistical Database Management 2004*

[16] Blythe, J, E Deelman, Y Gil, and C Kesselman. Transparent Grid Computing: A Knowledge-Based Approach. *15th Annual Conference on Innovative Applications of Artificial Intelligence (IAAI), August 12-15, Acapulco, Mexico. 2003.*

[17] Condor Website. Condor Cluster Middleware, Http://Cs.Wisc.Edu/Condor.

[18] DAGman Website. Condor Dagman, Http://Www.Cs.Wisc.Edu/Condor/Dagman/.

[19] Deelman, E, and A Chervenak. Data Management Challenges of Data-Intensive Scientific Workflows. *Proceedings of the 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid (CCGRID)* 687–92.

[20] Deelman, E. Pegasus: A Framework for Mapping Complex Scientific Workflows Onto Distributed Systems. *Scientific Programming* 13 (3/2005): 219-37.

[21] Fox, G, and D Gannon. Workflow in Grid Systems. *Concurrency and Computation: Practice and Experience* 18 (10): 1009-19.

[22] Freeflou. Freefluo Enactment Engine, Http://Freefluo.Sourceforge.Net/.

[23] Gil, Y, E Deelman, J Blythe, C Kesselman, and Tangmunarunkit H. Artificial Intelligence and Grids: Workflow Planning and Beyond. *Intelligent Systems, 2004, IEEE* 19 (1): 26–33.

[24] Gil, Y, E Deelman, M Ellisman, T Fahringer, and Fox G. Examining the Challenges of Scientific Workflows. IEEE *COMPUTER* 40, 2007, (12): 24–35.

[25] Glatard, T., J. Montagnat, D. Lingrand, and X. Pennec. Flexible and Efficient Workflow Deployement of Data-Intensive Applications on Grids With Moteur. *International Journal of High Performance Computing and Applications* 5 (20): 52-80, 2007.

[26] Gurmeet, Singh, Su Mei-Hui, Vahi Karan, Deelman Ewa, Berriman Bruce, Good John, Katz Daniel, and Mehta Gaurang. Workflow Task Clustering for Best Effort Systems With Pegasus. *MG '08: Proceedings of the 15th ACM Mardi Gras conference*

[27] Majithia, S., M. Shields, I. J. Taylor, and I. Wang. Triana: A Graphical Web Service Composition and Execution Toolkit. *IEEE International Conference on Web Services (ICWS'04)* 514.

[28] Miles, S., P. Groth, M. Branco, and L. Moreau. The Requirements of Using Provenance in E-Science Experiments. *J Grid Computing, 2007.*

[29] N, Mandal, Deelman E, Mehta G, Su M, and Vahi K. Integrating Existing Scientific Workflow Systems: The Kepler/Pegasus Example. *Proceedings of the 2nd workshop on Workflows in support of large-scale science, 2007.*

[30] Pan, M. J., D. Rex, and A. W. Toga. The Loni Pipeline Processing Environment: Improvements for Neuroimaging Analysis Research. *11th Annual Meeting of the Organization for Human Brain, 2005.*

[31] Ramakrishnan, A., G. Singh, H. Zhao, and E. Deelman. Scheduling Data-Intensive Workflows Onto Storage-Constrained Distributed Resources. *7th IEEE Symposium on Cluster Computing and Grid Computing (CCGRID)*

[32] Smith, M. G., and L. Bull. Genetic Programming With a Genetic Algorithm for Feature Construction and Selection. *Genetic Programming and Evolvable Machines, 2005.*

[33] Engine, Sun Grid. Http://Gridengine.Sunsource.Net.

[34] Surridge, M, S Taylor, and D Marvin. Grid Resources for Industrial Applications. *Proceedings of 2004 IEEE International Conference on Web.*