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: **D6.1b Distributed Medical Services Provision (Design Strategy)**

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.

1. Executive Summary

The aim of the neuGrid project is to provide a user-friendly grid-based e-infrastructure, which will enable the European neuroscience community to carry out research that is necessary for the study of degenerative brain diseases. The WP6 workpackage, Distributed Medical Services Provision, will supply a range of services to the users of the project. The provision of a group of generic medical services will enable Grid technologies to be applied in this and a number of other medical domains. This will provide the flexibility that is necessary for interfacing with existing medical systems and will allow the reuse of packaged services that exploit Grid functionality. This work package will gradually provide an Application Programming Interface (API) which is independent both from the application domain, the LORIS clinical data management tool which will be used in neuGrid and the underlying Grid infrastructure.

By abstracting Grid middleware specific considerations and customizations from clinical research applications, WP6 services should be able to provide generic functionality aimed specifically at medical applications. In the scope of this project, these generic services will be applied to satisfy the specific requirements of neuroscientists. They will bring together sources of data and computing elements into a single view as far as applications are concerned, making it possible to cope with centralised, distributed or hybrid data and provide native support for common medical file formats. Lower-level services will hide the peculiarities of a specific Grid technology from upper layers, thereby providing application independence and enabling the selection of 'fit-for-purpose' infrastructures. These *generic* services in WP6 will glue a wide range of user applications to the available Grid platforms thereby creating a foundation of cross-community and cross-platform services.

This work package will consequently identify services that can be made reusable for the medical domain. The output of WP6 will consist of a set of generic medical services which will include but will not be limited to query, workflow, provenance, glueing and abstraction, anonymization and portal services. Other services that are identified by the user requirements will be added where necessary. These services will enable the medical community and Grid community to work cohesively whilst maintaining their independence and generic behaviour.

The WP6 deliverable document outlines the design philosophy that is being followed during the construction of the distributed medical services and describes a design for the set of services that will constitute the distributed medical services layer. The design and evaluation process was led by user requirements, which have been separately elicited in WP9. The services have been designed by following a set of design principles whose features include:

- Services will be designed and built upon a service-oriented architecture.
- Services will follow a loosely coupled paradigm to reduce dependencies between components and services.
- Services will be made open and extensible so that new features and components can be added where necessary.
- Services will be exposed through interoperable interfaces and standard specifications will be employed and implemented where appropriate.
- Services will not be tied to a particular technology and will be middleware agnostic.
- Service reusability will be ensured to make them reusable and customizable across applications and domains.
- Services will be made as scalable as is architecturally possible to elegantly manage application and user loads.

Since requirement analysis is an evolving process and requirements may emerge as new functionality becomes available, the services will adapt to address the changes that are

necessary and will implement the functionality as and when requested by users. This extension and evolution mechanism has been established in the design philosophy and services can be seamlessly extended whenever a new feature is required. The requirements helped us to identify the following set of services that are necessary in the generic distributed services layer. The services will be designed in such a way that a variety of applications and Grid middleware can be supported.

- Pipeline Service: Using this service researcher can specify their workflows. This service will expose an API that can be used by workflow authoring environments such as LONI, Kepler, Taverna etc. to create and execute neuroimaging pipelines on Grid and distributed resources.
- Glueing Service: All the services in WP6 will access the distributed resources through a Glueing service. This service will provide an abstraction layer through which users can access data and other resources without lock-in to any particular middleware.
- Provenance Service: Provenance is the process of tracking the origins of data and its evolution between different stages and services. This service will capture, store and perform analysis on the data for better decision making.
- Querying Service: The querying service will be a generic service that can query and browse data that is stored in a file or relational database or in other Grid databases. The query service will provide an easy to use and high-level querying mechanism that is independent from the underlying data resources.
- Portal Service: The user will interact with the system through a portal service. This will be the point of entry into the system and all other services will be accessed through this service. It will also hide the low-level service interfaces and implementation details from common users.
- Anonymization Service: Biomedical data needs to be anonymized and should be shared after ethical and legal clearance. The anonymization service will ensure this before the users can access biomedical data for their analysis. This is a generic service and can be reused in other medical projects and applications that have similar requirements.

In the design document, an effort has been made to produce designs of these services that can best address the user requirements (as identified in workpackage WP9) and at the same time are consistent with the design principles that were summarized earlier. The following sets of activities were performed in order to produce the services designs:

- An analysis of the requirements was carried out to identify the components that can best address the user requirements. A group of generic components that can address these and other requirements that may emerge in future was created.
- Components were identified that provide common features or have overlaps with the functionality required. All such items were bundled together to produce a group of components that can address an area of the requirements. The grouped components were exposed as the candidate services that have been described in the above sections.
- A service design was produced for each of the services using the defined design philosophy. This design process not only glued together various components, it also ensured that these components and services can be extended as and when required. It was also ensured that minimum dependencies existed between components and services and they were sufficiently scalable to cope with future demands in loads. Particular care was taken to make these services as generic as possible and vendor and middleware lock-ins were avoided.
- Suitable interfaces were crafted to provide access to these services. It was ensured that these interfaces promoted interoperability and ease-of-use. Standard

approaches were employed in designing these interfaces with the intention that services should follow generally agreed standards to the greatest possible extent in order to make them widely exploitable.

- An analysis and evaluation process was initiated to investigate emerging technologies and tools that could help in producing the features that were needed by services. This process helped in identifying a set of technologies that could be reused in implementing the services. This activity also produced a list of functionality in each of the services that could not be implemented using existing technologies and where new development would have to be carried out. This process also produced a comparative analysis of the state of the art in each of the services and identified missing functionality.
- The WP6 design has delivered a set of services as presented in this document. It has also considered and described how state of the art technologies can implement these services and considered the advantages and disadvantages of each. Recommendations are made regarding the use of interfaces and suitable technologies. Any missing functionality is described and a roadmap for implementing and delivering the services that have been identified has also been set out (The detailed design specifications will be submitted in the next couple of weeks.)
- Each service describes the areas in which more research is required and states what can be quickly integrated to release an initial set of functionality in the form of prototype systems. An effort will be made to deliver initial functionality in each of these services in year two and further research will be carried out in the following years to add additional functionality.

This first version of the WP6 document follows the requirements analysis process, while delivering early insights on possible solutions. It is anticipated that the document will have a series of refinement iterations in light of the requirements analysis delivery (Month 14) and throughout the neuGRID platform development. In the following sections, the reader will gain an understanding of the approach that is driving WP6 and its developments. The document establishes the features which the solution should support, and as such, it is intended to be a useful reference throughout the development and testing phase of the neuGrid prototype.

2. Introduction

The medical domain is increasingly becoming more compute and data intensive leading to the adoption of distributed computing infrastructures. In order to facilitate analysis, querying and collaboration over such infrastructures, numerous efforts in the domain have built community specific high-level distributed services such as querying, workflow management, provenance, image handling and anonymization services [1][2][3]. Such services help medical users in sharing data and knowledge and enrich medical decision support systems. Most of these services are DICOM-aware Grid services designed and developed for a particular community of medical users. Services are often built specifically by one community and cannot therefore be readily shared and re-used in other medical domains due to architecture, interface and platform limitations. WP6 aims to develop generic services based on a Service Oriented Architecture (SOA) philosophy. The services, developed as part of WP6, will be derived from the neuGrid project user requirements as presented in section 3. However, because there is an overlap in terms of the functionality required from a Grid or any distributed computing infrastructure in multiple medical applications, WP6 will focus on designing *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 section 2 in order to enhance and promote

their re-usability in other related applications. In the scope of neuGrid project, the services will be implemented and demonstrated on top of the g-Lite middleware.

In order to enhance the reusability of the services developed in WP6 one of the major design considerations is to develop them in a manner that makes them independent from the underlying Grid middleware. This will increase the flexibility of the medical services layer and allow it to make use of emerging advances in Grid technologies with minimum changes to service interfaces. The services should therefore be designed and implemented in such a way that users do not require a high-level of Grid knowhow. The underlying technology implementation details should be transparent from the users. This is clearly beneficial since users often find it difficult to cope with the inherent complexities of such infrastructures. As figure 1 illustrates, the WP6 distributed medical services will sit between the Grid services and the user facing tools. The Grid will act like a "black box" for the WP6 services and suitable services will be crafted to provide the required abstraction from the Grid resources. The WP6 distributed medical services will access Grid services through the abstraction layer in a transparent fashion in order to avoid lock-in to a particular Grid middleware. This loose coupling philosophy between the Grid and generic services will be a fundamental design principle in WP6.

By abstracting Grid middleware specific considerations and customizations from clinical research applications, WP6 services should be able to provide generic functionality aimed specifically at medical applications. In the scope of this project, these generic services will be applied to satisfy the specific requirements of neuroscientists. They will bring together available sources of data and computing elements into a single view as far as applications are concerned, making it possible to cope with centralised, distributed, or hybrid data, and provide native support for common medical file formats. The lower-level services will hide the peculiarities of a specific Grid technology from the upper layers thereby providing application independence and enabling the selection of 'fit-for-purpose' infrastructures. These *generic* services in WP6 will glue a wide range of user applications to the available Grid platforms thus creating a foundation of cross-community and cross-platform services. A schematic diagram of this layered approach is shown in figure 1.

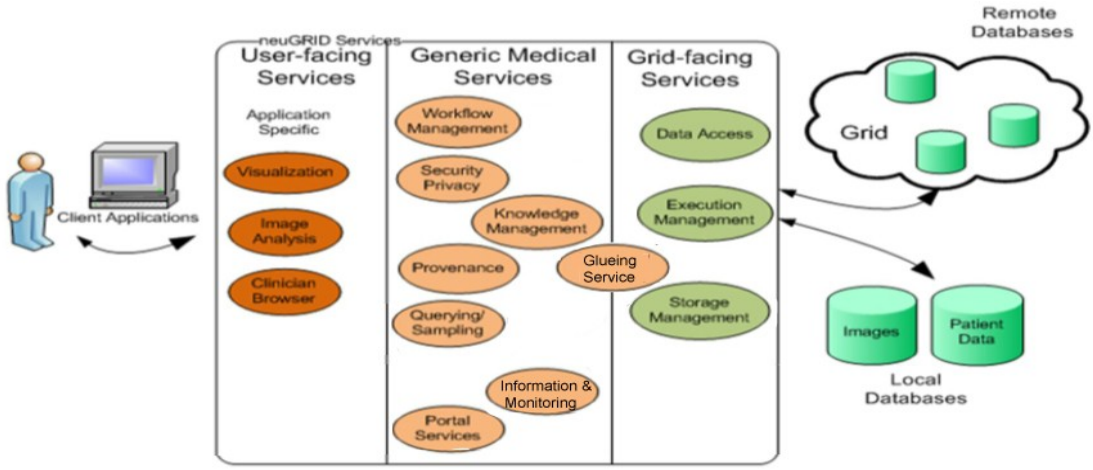

Figure 1: Layered Services Architecture

The Distributed Medical Services Provision in WP6, will take care of supplying generic and cross community services to users. This work package will therefore identify a group of generic services that widely applicable in the medical domain. These include medical query

and image handling, workflow, provenance, access control and algorithm distribution. The provision of generic medical services to enable Grid access from a number of medical application domains will provide flexibility for interfacing further existing medical systems, reuse of packaged services and exploiting Grid functionality for a wide community of clinicians. This work package will provide an Application Programming Interface (API) that is independent both from the Prodema (PR) application domain and the underlying Grid infrastructure. These services will sit between PR services (WP5) and Grid-specific services (WP7) to enable this independence.

The generic medical services that will be developed may include query, workflow, provenance, glueing, anonymization and privacy elements. Other services that are identified by the user requirements will be added where necessary. These services will enable the medical community and Grid community to work cohesively while maintaining their independence and generic behaviour. Initially, the top-level user-facing services (WP5 Brain Imaging Services) will be tightly coupled with the lower-level grid-facing services (WP7 Grid Services). Using the recommendations for standard medical Grid services, we will start to factor out functionality from each side into the middle layer. It may be that some of the existing services implemented on each side are sufficiently generic to be included in this layer with little modification, while others will require an extra abstraction layer to be inserted between them.

The aims of this work package therefore are:

- To produce a set of generic medical services to sit between the user-facing services that are being developed in WP5 and the Grid Services being provided by WP7.
- To ensure that sufficient functionality is provided to address the needs of neuroscientists' for the analysis of Alzheimer's disease.
- To provide independence from any specific Grid-technologies that are available for use in medical Grids.
- To provide a flexible and re-usable set of medical services that follow the SHARE recommendations [4] and that can be extendable to other medical domains in the future.

3. The Services Design Philosophy

In the following sections, we describe the fundamental principles of the design philosophy. The distributed services in WP6 will be built and delivered by following these guidelines. After the design philosophy has been discussed, we will present a list of the services that will drive the WP6 effort. The need and role of these services will be justified through the analysis of the user requirements. The neuGrid project requirements have already been elicited and discussed in WP9.

3.1 The Services will follow a Service Oriented Architecture

A particular problem with Grid applications is to make them flexible enough to be used across a range of potential platforms and environments. While previous Grid architectures used a dedicated solution with rigidly controlled hardware and environments, it has recently become clear that making Grid applications run on a wider range of platforms enables users to easily expand the scope and power of the Grid. However, minor differences between platforms can cause significant headaches. An obvious solution is to remove the highly platform-specific elements and move to a more generalized environment. A widely used and

standard form of a generalized distributed environment which can support a wide range of applications and platforms is a service oriented architecture (SOA). Grid architectures are distributed in nature, hence the SOA paradigm is a natural fit. The medical services layer is planned to be implemented using the SOA paradigm, firstly in order to leverage the advantages mentioned above and secondly to have a potentially flexible and reusable medical services layer which can be customized for various applications in future.

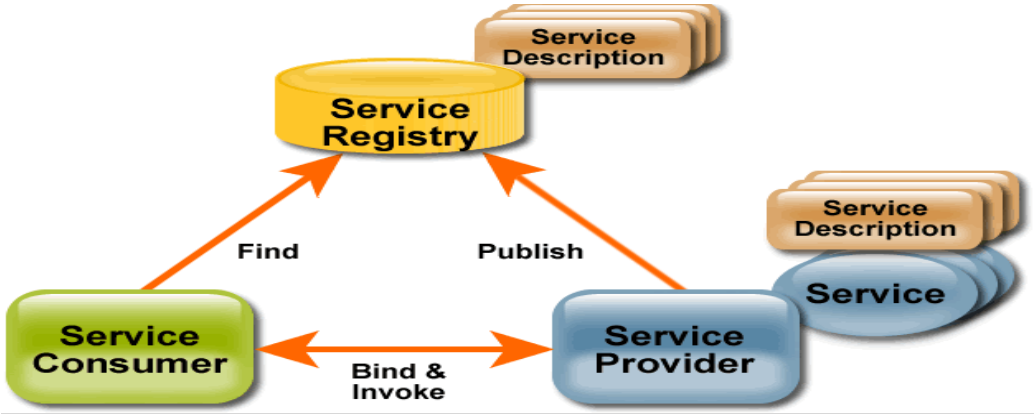

Figure 2: SOA Conceptual model

A Service Oriented Architecture (SOA) is a loosely-coupled architecture for software and has several definitions [5]. The simplest SOA consists of three basic elements: service producers, service brokers and service consumers. As shown in figure 2, the service provider's role is to deploy a service on a server and to generate the description of this service (the service contract), which defines available operations as well as invocation mode(s). This description is published in a directory of services inside a service broker. (This is shown as service registry in figure 2). Thus, service consumers are able to discover available services and obtain their description by interacting with the service registry. The obtained descriptions can then be used to establish a connection with the provider and to invoke the desired service operations. A SOA enables the flexible integration of applications and resources by: (1) representing every application or resource as a service with standardized interface, (2) enabling the service to exchange structured information (messages, documents, objects), and (3) coordinating and mediating between the services to ensure they can be invoked, used and changed effectively.

Due to the loosely coupled and flexible nature of the architecture, new components can be easily added with little or no alteration to existing component interfaces. The self-contained modular structure of the components also enhances reusability of the components. As shown in figure 3, by putting the logic into a separate layer, the layer can exist well beyond the lifetime of any system it is composed into. From higher quality to more reuse to better scalability and availability, the benefits of implementing a service layer far outweigh the cost and additional effort involved in its implementation. Components of software developed according to a SOA are distributed, self-contained and modular services. The distributed and self-contained features of modules make this architecture fault tolerant. Individual components may be susceptible to faults but consumers can discover replica services for their operations via service registries. Services in SOA interact via standardized interfaces, for instance the Simple Object Access Protocol (SOAP) is used as a standard communication protocol between web services. Because standardized interfaces are used for interaction amongst services, disparate parts of the software can be developed for, and deployed on, any platform. SOA is based on a combination of the principles of object orientation and Web services and also marries this structure with an open system describing the interfaces available. By making Web services easier to find and identify, SOA makes it easier to deploy and distribute a SOA-based application. Because Web services are based on open standards

and are, by definition, architecture- and platform-neutral, SOA-based applications can be deployed across a wide range of platforms.

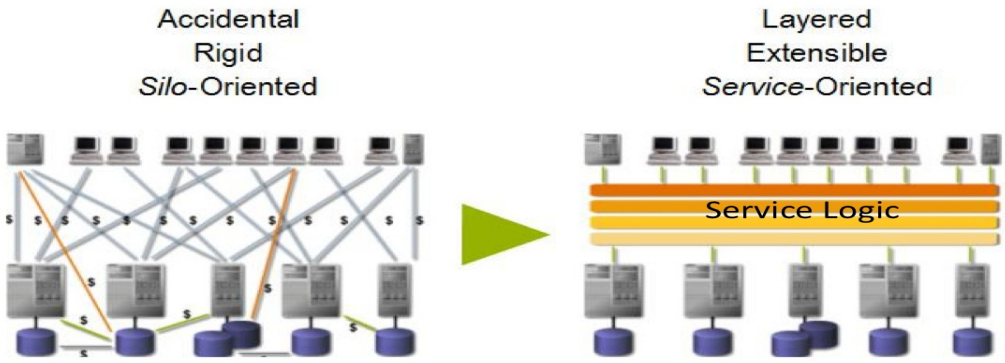

Figure 3: The "Move" to Service Orientation (source Sun Microsystems)

Location transparency and service reuse are the properties of a service-oriented architecture which help in code and service mobility. To achieve location transparency, applications look up services in a registry and bind to them dynamically at run-time. This feature promotes scalability since a load-balancer may forward requests to multiple service instances without the knowledge of the service client. The lookup and dynamic binding to a service means that the client does not care where the service is located. Consequently, users have the flexibility to move services to different machines and hosts. Location transparency also enables services to remain highly available. Because of location transparency, multiple servers may have multiple instances of a service running on them. If a network segment or a machine goes down, a dispatcher can redirect requests to another service without the client's knowledge.

Service oriented architectures tend to improve and facilitate service maintenance. By focusing on the service layer as the location for core business logic, maintainability increases since developers can more easily locate and correct defects in the services. Moreover, services have published interfaces that can be easily tested and validated independently from any application that uses the service. In the end a service oriented architecture provides a library/catalogue of services that are well tested, have well defined interfaces, are platform and language neutral and are properly validated so that they are ready to be used in different applications to be deployed across a wider set of platforms and domains. Grids therefore have a lot to gain from the SOA philosophy. Initial Grid middleware infrastructures were not SOA based and faced numerous challenges related to scalability, flexibility and robustness. SOAs offer flexible methods for adjusting the architecture of Grid solutions and making them more transparent and supported on a wider range platforms and environments. The SOA Grid can, however, be susceptible to problems such as latency, concurrency and partial failures.

The whole discussion suggests that a SOA could be a viable distributed environment to design and build generic medical services. In neuGrid Services in a SOA are built via standardized components and interfaces. Standardized interfaces increase the potential for the re-use of services in a range of configurations. This enables the system to provide features which the architecture was not originally designed to support. For example the Pipeline Service currently is designed for neuro-imaging pipelines, but the same service can be used for any other application in any environment as long as the standard interfaces are (re-)used. The architecture itself is flexible, new services can be developed in future and embedded into the architecture with little or no modification to any other services, as they are independent, modular and loosely coupled. Finally because of standardized interfaces, the internals of the service can be developed on any platform, providing a platform and middleware agnostic solution.

3.2 The Services will be loosely coupled

Coupling is a measure of the dependency and commonality required by a service user and service provider to be able to achieve a result [6]. The ability to compose dynamic services in environments of long-running jobs and transactions requires a certain measure of loose coupling. However, there are different notions of "loose coupling" and the ways it can be implemented. A service is more loosely coupled than others if it allows for a greater degree of variability and freedom for that particular aspect. The commonly adopted approach though is to introduce a minimum set of dependencies between services in order to support better the reusability of existing components (i.e. services which are already deployed). Moreover, these services should be combined in order to quickly and cost-efficiently respond to new demands. To achieve this goal, some engineering rules which are not always specific to SOA, have been identified that guide the loose coupling behaviour of services.

In a SOA, a service client should not be implemented to interact directly with a particular service producer. Any change to the service producer would require commensurate changes to all clients if they communicated in such a direct fashion, making them tightly coupled. This RPC-style approach loses the real service-oriented nature of a service. A service is meant to be an independent entity that users can compose into workflow processes as they desire. In a SOA, it is the service consumer, rather than the service producer, that defines how an application behaves. Yet, loose coupling mandates that service consumers and service producers be independently created and controlled. Given the highly variable nature of the services being exposed, service consumers depend on registries in order to find and bind to the appropriate service. Dynamic discovery and the associated routing enables the association of service requests with the appropriate services at runtime, regardless of any API-specific characteristics of the underlying components. As a result, it is clear to see that in a truly loosely-coupled SOA world, it is as impossible to operate without a registry as it is to operate in a global Internet without a DNS (the Domain Name System which establishes a hierarchical but decentralized relationship between human-readable names and IP addresses).

We conclude this section by summarizing that the medical services in WP6 will be made loosely coupled. Some of the major features that these services will provide include the following:

- The services will be made flexible to compose them into adaptable workflows; they will be dynamically discoverable and components dependencies will be minimized.
- An attempt will be made to make the services loosely coupled at the specification and implementation level.
- It will be ensured that the service contract remains the same for the lifetime of the project. Generic interfaces will be defined that can be extended if new features are required.
- The service roles and behaviour will be mapped to number of WSDL documents.
- The services contracts and implementations will be abstracted from users and only minimum possible information will be exposed.

3.3 The Services will be open and extensible

Service oriented architectures promise a standards-based platform for distributed computing focused on simplicity and flexibility [7]. One key service technology is WSDL, the Web Service Description Language. Using WSDL, architects can describe their Web services in abstract form, similar to existing Interface Description Languages (IDLs) [8] used

in other distributed computing frameworks such as CORBA [9]. WSDL also allows Web service developers to specify concrete bindings to a service. As shown in figure 4, these bindings describe how the abstract service description is mapped to a specific access protocol. This portion of WSDL is extensible, which means that anybody can come up with their own binding such that it may be possible to access the service through some customized protocol. There can be multiple bindings to the same service. The binding itself may represent a mapping of the abstract service description to any arbitrary protocol for accessing the service. Having multiple choices in the way the service is used and allowing binding extensions is useful, since it makes it possible for clients to conveniently access and compose the services in different application scenarios.

WSDL allows users to create open and extensible services by providing a mechanism by which multiple service interfaces and bindings can be defined. WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. As shown in figure 4, the operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related endpoints are combined into abstract endpoints (services). WSDL is extensible to allow the description of endpoints and their messages regardless of which message formats or network protocols are used to communicate. Architects and developers have a choice of using either RPC or document style messaging and are able to use the right technology for the tasks they face. Good design dictates that the method signature of an RPC message service should never change. With document messaging, the rules are less rigid and many enhancements and changes can be made to the XML schema without breaking the calling application.

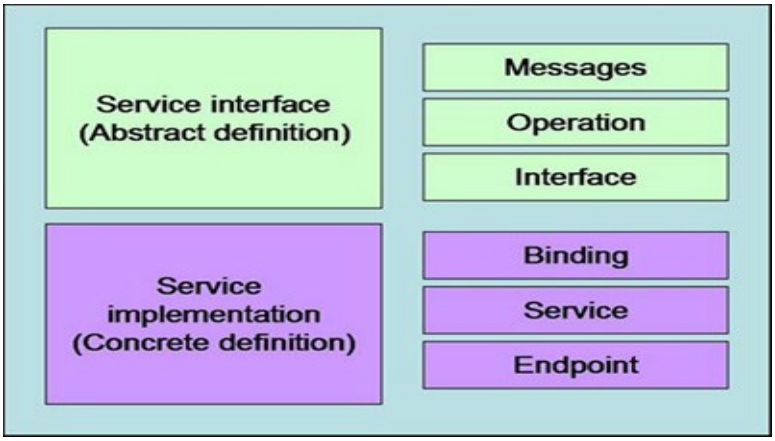

Figure 4: WSDL Conceptual model

Clients that want to exploit the availability of multiple bindings would then have to possess the ability of switching the actual binding to be used based on run-time information. So in order to take advantage of Web services that offer multiple bindings, clients need a service invocation mechanism that allows them to switch between the available service bindings at runtime, without having to generate or recompile a stub. For example, imagine that you have successfully deployed an application that uses a Web service offering multiple bindings. To make this example more concrete, suppose you have a SOAP binding for the service and a local Java binding that allows you to treat the local service implementation (a Java class) as a Web service. Obviously, the local Java binding for the service can only be used if the client is deployed in the same environment as the service itself, and, if this is indeed the case, it is far more efficient to communicate with the service by making direct Java calls rather than using the SOAP binding. The Java binding acts as a kind of shortcut access mechanism.

A WSDL description for each of the WP6 services will be created. This will not only describe a service, its endpoints and operations, but it will also describe the interfaces and features. Therefore,

- The services in WP6 will be described through WSDL (ie. the operations (or methods) the service exposes). All the interfaces and bindings will be defined and specified through their respective WSDL's.
- WSDL will be a basis of contract and communication between different services as well as with clients
- The services in other work packages and clients will use WSDL to access the WP6 services, their methods and functionality
- WSDL will be used to locate the services
- WSDL will be used to access and support multiple bindings

3.4 The Services will be interoperable and will follow generally agreed standards

A service-oriented architecture (SOA) depends on a minimum number of interdependencies between services. This loose-coupling should propagate all the way to the protocol layer of the architecture. Thus communication infrastructure used within a SOA should be designed to be independent of the underlying protocol layer (as shown in figure 5). Designing an effective distributed software system becomes especially difficult when considering the fact that services and processes might need to interact with each other across multiple protocols. Code reusability becomes one of the most important benefits derived from a well-designed protocol-independent communication framework, since the code that uses the framework will not be tied to the protocols used. Transparent replacement of the underlying protocol is also a very important benefit derived from a protocol-independent design. This facilitates deployment of new protocols without the typical redeployment headaches such as server restarts and frustrated users.

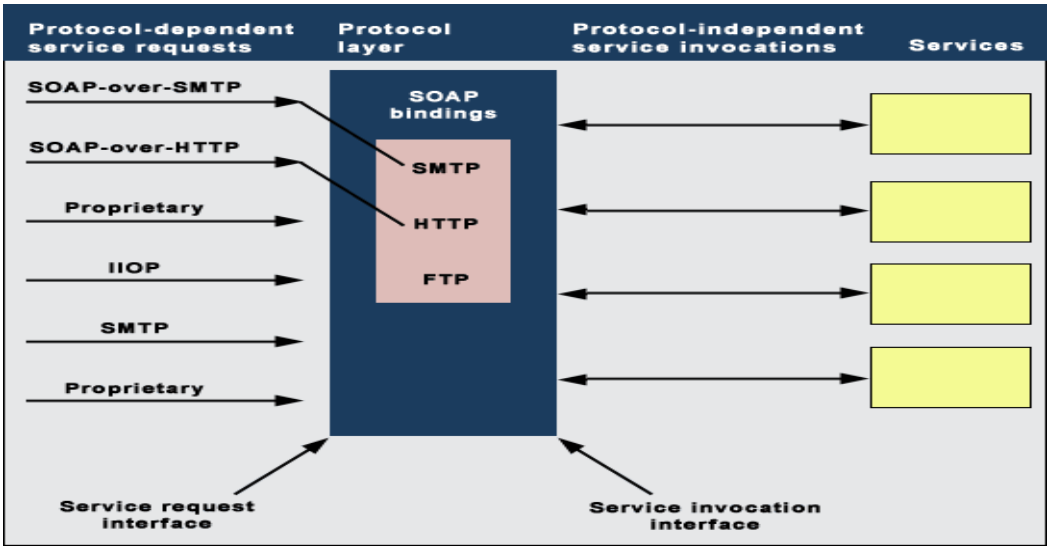

Figure 5: SOAP and the protocol abstraction layer (Jeff Hanson, 2007 Techrepublic.com)

Encapsulation and abstraction principles originally came from the world of object-orientation. The idea was to hide self-contained information of a service from end-users and to propose only one stable interface exposing the details considered to be necessary for handling it. A

service is therefore seen as a black box from the outside, which makes it possible to separate its interface (its external description) from its actual implementation. One can thus modify a service implementation without changing its interface, which turns it into a sustainable model. Messages delivered by a service should not contain business logic. On the contrary, they must be restricted to the transport of, and only of, data structures from one service to another. That makes it possible to modify or to add services without impacting the other services of the architecture. These aspects are addressed thanks to different specifications such as WS-Enumeration [10], SOAP-attachments [11], WS-Notification [12], etc but the interoperability and message communication through standards protocols needs to be tackled right from the service inception and design.

The Simple Object Access Protocol (SOAP) has become almost synonymous with web services, even though it is just one of many possible ways to access them. This means that applications that make use of web services usually do so through APIs tied to a specific implementation of SOAP. But there needs to be a mechanism to describe and access the services in a generic way. SOAP is a lightweight, XML-based protocol for exchanging information in a decentralized, distributed environment. SOAP supports a Remote Procedure Call (RPC) style of information exchange and message-oriented style of information exchange. SOAP is designed around a number of loose-couplings including protocol independence, language independence, platform independence, and operating system independence. SOAP version 1.2 messages can be transported across HTTP, SMTP, or any other protocol for which a binding conforms to the binding framework. The SOAP Binding framework can be easily integrated with existing protocol abstraction layers by providing bindings for protocols that are already handled in the legacy code. The services in WP6 however will be designed to support any communication protocol and messaging mechanism, but for the scope of the project these services will use the SOAP protocol.

In conclusion, the WP6 medical services will be guided by the following design principles to make the services protocol independent:

- The design and implementation of the services will be protocol independent. (Means will be provided to describe and access the services in a generic way);
- The services will be accessed and communicate through the SOAP protocol;

- The services will be interoperable. For the scope of the project, the SOAP protocol will be used to achieve service interoperability;
- Mechanisms will be put in place for code reuse and plug and play communication through any other protocol during the service deployment and operational life

3.5 The Services will be Grid middleware agnostic

The medical domain is increasingly building high-level distributed services such as querying, workflow, meta-scheduling, image handling and repository services. Such services help medical users to analyse their data, extract knowledge from it and share the results with other users. Most of these services are designed and developed for a particular community of medical users. Not only that, these services are also developed for and deployed on a particular Grid middleware. It is very rare to find services that may serve the needs of another community and can run on a middleware other than the one for which these services were created. The services built by one community often cannot be shared or re-used in other medical domains due to architecture, interface or platform limitations. The services created in WP6 will be middleware agnostic and will shield the heterogeneity of distributed resources through a common abstraction layer. This will enable the services to access Grid and non-Grid resources alike without getting locked in to a particular Grid middleware. The Grid is

evolving and new features are being added to Grid middleware every now and then. The service interfaces need changes with every new release of a middleware to cope with the middleware evolution. The middleware agnostic services will not have to undergo these changes since an abstraction layer will shield the services when the underlying middleware evolves.

For example, consider a service being deployed on g-Lite and a user wanting to use a different Grid middleware since he cannot find some features in the available middleware. In the current scenario, this transition from g-Lite to any other middleware is not straightforward and needs quite a lot of changes in the code, recompilation and redeployment on the new middleware. This is not a good approach since this takes away a user's attention from an application to the middleware functionality and as a result, users are less interested in porting the applications to Grid. If a mechanism is developed where users are not concerned about the Grid fabric and functionality and at the same time all the Grid details remain abstracted from users, this will help not only to make the Grid use ubiquitous but will also make the life of the users easier. Applications and services developed for one platform can then be deployed on any other Grid middleware without the current effort and major rework that is often required at present to achieve this task. This is one of the core design objectives of WP6 where generic services will be developed that can run on any middleware and vendor lock-in can be avoided.

Despite the progress on Grid technologies, Grid applications are far from ubiquitous, and new applications require a significant degree of programming effort just to access the Grid resources. There have been developments at various standards forums that may help us to address these issues. A key impediment to an accelerated deployment of Grid applications is the scarcity of high-level application programming abstractions that bridge the gap between existing Grid middleware and application-level needs. Recently there have been some developments which make this abstraction easier. The Simple API for Grid Applications (SAGA) is an OGF standardization effort that addresses this particular gap by providing a simple, stable, and uniform programming interface that integrates the most common Grid programming abstractions. Figure 6 depicts the use of the Glueing services to abstract the Grid middleware and shield the services from the underlying Grid technologies. The Glueing service uses a SAGA implementation to provide the abstraction that is required to shield higher level medical services from the underlying Grid middleware. In figure 6, more than one application has been shown to access the resources provided by different Grid middleware. We will try to provide features such as workflow management, task dependencies, security and context management etc. which are not yet supported by different SAGA implementations, through the Glueing service. The Glueing service will only implement a new feature if this is not available in a SAGA implementation; otherwise an attempt will be made to use as many features from SAGA implementations as is possible to promote standard access and ubiquitous interfaces.

The high-level interfaces, provided by the Glueing Service (as shown in figure 6), for communicating with different Grid middleware minimize the need to use middleware specific APIs. For example, if an application intends to use three different middlewares then the application has to use a different API set, provided by the middleware, for interacting with each middleware. SAGA, on the other hand, provides generic API calls to interact with every middleware whose adaptor is loaded in the application. Thus, SAGA facilitates the glueing of a number of client applications with different middleware using the respective middleware adaptors.
To conclude this section, middleware agnostic medical services will be designed and implemented in WP6. The SAGA API, implementations and adapters will be exploited to provide the desired abstraction. In this regard, the salient design features of the WP6 services include the following:

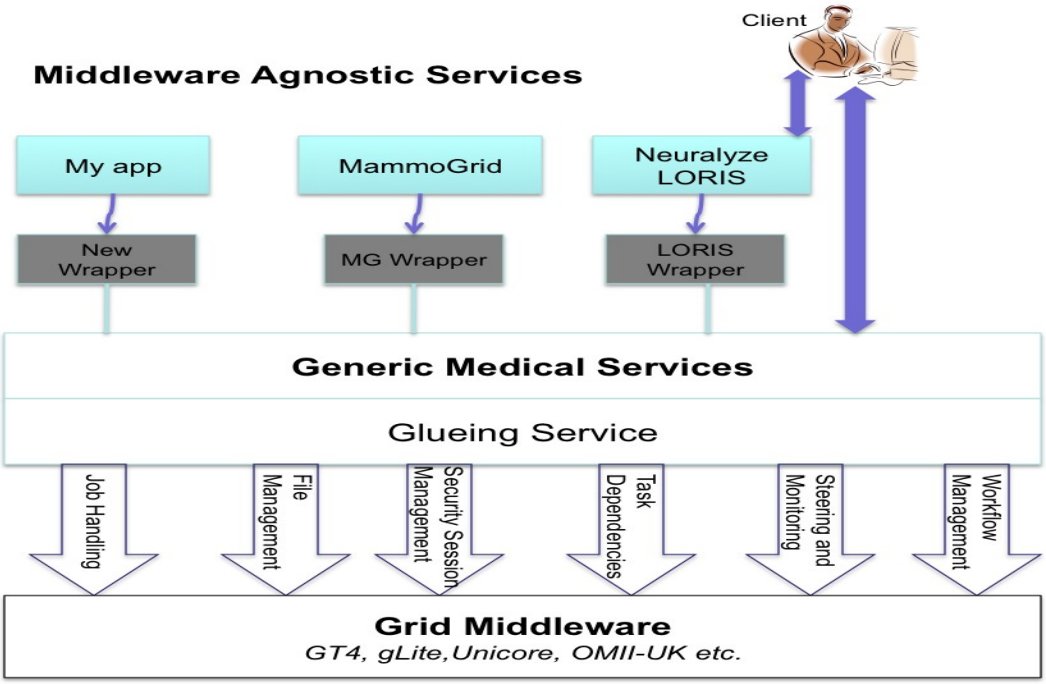

Figure 6: Middleware Agnostic Services

- The services will not be tied down to a particular middleware
- The services could be deployed on any Grid middleware including the gLite middleware
- The services will be adaptive; services can access a wide range of middleware at run time
- The services will use the SAGA specification and the Glueing service to abstract the middleware functionality.

3.6 The services will be reusable

The scientific community in general and the medical community in particular has been developing a number of services to address their analysis problems. These services are developed normally for a particular environment that cannot be easily ported and transformed to other scientific communities. Services developed in one language cannot talk to a service in any other language even if they are using standard protocols. Similarly, services deployed on one middleware platform cannot run on another platform due to interface, architecture and protocol limitations. The objective in WP6 is to design and develop services that can be reused in a wide variety of applications and projects. A service developed by one community should be easily adapted in other projects without any major change in the code and interfaces. The services should talk to each other without major restructuring as far as they use standard protocols. Even if they are deployed on different operating systems this should not matter since they use a common messaging and communication mechanism.

An elegant interface is the essence of a good service design. Combined with the use of standards, interfaces are the essential ingredients for creating a loosely coupled system where service clients and service providers can communicate regardless of the programming language and platform. Services are to be independent, in that clients need not understand the inner workings of a service component; essentially the service operates as a "black box." "White box" reuse, or cut and paste, where source code is modified in order to use in another context, while useful, is not typically as beneficial as "black box reuse." Systematic reuse

programs encourage reusing software without change because of the benefit they receive from black-box reuse throughout the life cycle. Domain analysis is a means to determine the commonality that exists across the domain, and thus determining what types of services would have reuse value. Domain analysis can be achieved in a number of ways. With a reuse engineering model, it occurs one project at a time. Researchers work with project teams to capture the capabilities that project teams need to deliver. Analysis of this information in a given domain reveals where the duplication of effort exists and where services could be introduced to create reuse value.

Services must also be designed in a format which enables composition of services in workflows. Although service composition in a form is reuse, there are important differences which must be considered in the design. A reusable service is a service which can be reused in a number of ways, including different applications, however the service consumers are clients. A composable service however, is a service which can be used in workflows where the service consumers are services themselves. Hence in order to build composable services interfaces must be designed in a format which make them compatible with other services. Not all services are suitable for composition into workflows. Therefore, service reusability and composition should be addressed at the same time in the service design.

To conclude the section, the medical services in WP6 will be reusable and composable. The salient features may include the following:

- The services will be made reusable across domains and platforms;
- The services will be made language and platform independent by adopting standard protocols;
- The services Blackbox reuse approach will be preferred over the Whitebox approach of reusability;
- The services interfaces will be designed to achieve true reusability;
- The services will be made composable to be used in different medical applications;
- The services reusability will address the service evolvability and agility needs;

3.7 The services will be scalable

Services need scalability for handling increased loads. Scalability provision in the architecture not only ensures service reliability and high availability, it also improves performance by reducing the associated latencies and improving the access times. Problems encountered in managing service scalability are similar to those in scaling the distributed infrastructure in general. There are a number of ways to scale the services and WP6 will be made scalable by the use of the following possible approaches.

A well-formed scalable service needs to be stateless. Stateful services, although providing a better quality of service, are less scalable due to the inherent tight coupling involved and additional state management constraints. This rule, which can seem very constraining, must be moderated. It is generally accepted that the state conservation (i.e. the management of the context) as well as the action coordination (i.e. the management of the transactions) are localised in a specific function of the SOA, such as the service composition and orchestration which are application specific and may change from one application to another. The application of such a rule facilitates the reuse, the scalability and the robustness of services. Cohesion is a delicate rule to define. It translates the degree of operations and functional proximity inside a service. In other words, it aims at facilitating the comprehension and reusability of a service by gathering homogeneous operations corresponding to similar business logic. A scalable service should be idempotent, which makes it possible to be unaware of multiple receptions of the same request. The idea is that the use of such a service makes it possible to slacken the assumptions of reliability on the communication layer.

Redundancy is another way to cope with increasing loads. Multiple instances of the same service are deployed and through load distribution methods, the load is distributed between the service instances to achieve scalability. Scalability in network-available services, which might be defined as the ability of an application to handle growth efficiently, is typically achieved by making them available on multiple devices. Whilst a single server can be enlarged to a certain degree, this approach rapidly reaches a point where the cost of scaling overrides the benefits. To function on a number of instances, an application needs to do two things; direct requests to an appropriate server, and enable that server to process them and provide an appropriate response. In this model, service request messages are sent to a service's URI, but some mechanism (either in the message or external to it) routes them to another intermediary device. There are many methods of directing requests to a service intermediary, depending on the nature of the deployment and the service's requirements. The XML Protocol [13], or an XMLP Module, may provide a mechanism for routing messages to the device. Alternatively, if a number of service intermediaries are located (network) near each other, a "Layer-3+" load-balancing switch may be used to distribute the load between them, without explicit in-message routing. In a more distributed deployment, a "Global" load balancing service may be used to direct clients to the appropriate service intermediary based on some criteria, and achieved through any of a number of possible techniques acting at various layers.

Another mechanism that can be used to scale the services is the use of caching techniques. Caching is a technique that has been used for some time to scale distributed systems, whether it be in file system design or the World Wide Web. By allowing clients to keep and reuse copies of entities, efficiencies are realised by either the avoidance of data transfer, or the avoidance of a round-trip to the server altogether. Caching techniques rely on locality in usage patterns; that is, the likelihood that portions of messages can be reused. Some services consist of the submission of a message as the request, and a brief acknowledgement as a response, in a manner similar to SMTP's store-and-forward pattern. Standardization of an acknowledgement message would allow intermediaries to take responsibility for handling requests whilst immediately acknowledging them. In combination with caching and other techniques, store-and-forward allows intermediaries to improve service reliability substantially, by making it possible to have multiple, redundant points of contact for message submission, with the possibility of performance improvement through client/intermediary locality. In some situations, intermediaries need to send or receive a number of separate messages to or from a particular device. Although some transport bindings may make it possible to reuse a network connection for these messages, further processing efficiencies might be realised by their combination into a single message. For example, it might be desirable to send all store-and-forward messages for a service at once, wrapping all of them in a master message which uses an encryption module to protect them. If used across an HTTP binding, this approach avoids the overhead of separately encrypting the messages and then submitting each one and waiting for a response to indicate success.

To conclude the section, the medical services in WP6 will be made:

- Scalable and light-weight by creating multiple instances of the services that may communicate with each other;
- Load-aware by using load balancing techniques;
- Stateless and idempotent to achieve scalability;
- Reliable with the use of service intermediaries;
- Highly available by adding redundancies;
- Performance aware with the help of caching strategies and
- Efficient by grouping the messages.

4. The neuGrid project requirements

The following paragraphs in this section list the major requirements (that have been collected in WP9) that the users are expecting to be addressed in the neuGrid System. Only those requirements that come under the scope of WP6 and need to be addressed in the services layer will be discussed here. A candidate service in WP6 will implement the mandatory requirements related to that service and optional but related requirements will be addressed as additional features of these services. Consequently, the set of services that will emerge from the user requirements will be discussed in this section and their interaction with the rest of the system will be illustrated. The invocation of and association among these services and how they will be formed into a functional neuGrid system will also be discussed. After the requirements and a brief description of each of the services have been discussed, a tentative design and architecture of each of these services will be outlined in the later sections of this document.

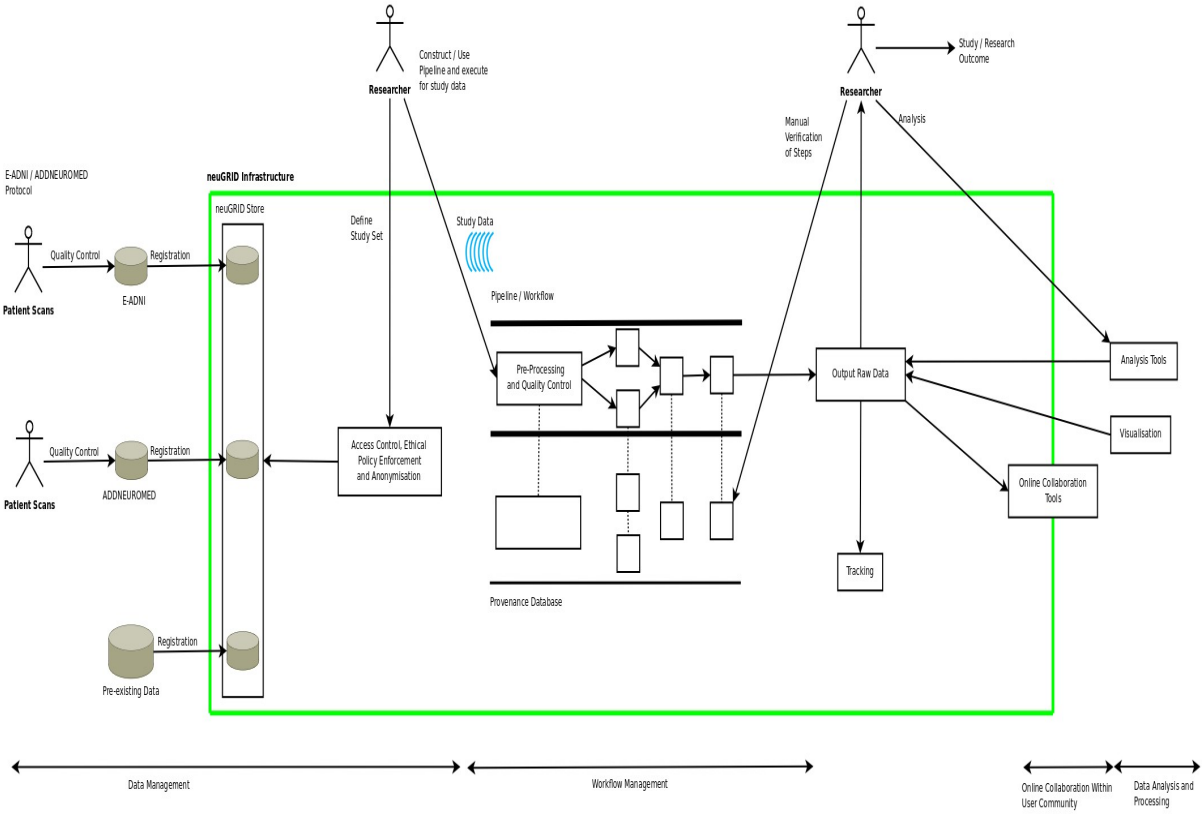

Figure 7: An end to end example of the neuGrid architectural model

In the neuGrid project, a user is expected to pass through the following stages (see figure 7) to carry out his or her analysis on a set of images:

- Data registration into the neuGrid Store, data management and quality control;
- Data access, querying and browsing;
- Workflow development, execution and management;
- Validation of results and workflows using the provenance data;
- Sharing workflows, histories and results and
- Visualization of the results.

The first stage in the analysis cycle is to register images in the neuGrid store that have been collected from the hospital data acquisition system or have been imported from other research

projects. For example, a new clinical site may wish to make use of the neuGrid infrastructure to share data within the wider research community. Existing data is thus put through a process which enforces quality control, formatting and ethical compliance. Finally the data is integrated with the neuGrid standard data model, which enables other researchers to access it and carry out their research. As new data sets are acquired they go through an initial local quality control step before passing through the same system-wide quality control, formatting, ethical compliance and data model integration processes that the pre-existing data goes through.

The role of second stage in the analysis process is to make the data browsable through automated querying tools. Therefore, an appropriate data access mechanism needs to be put in place. For example, a researcher may be interested in a rare form of a disease and wants to do a statistically meaningful analysis. Unfortunately the researcher's institution does not have sufficient images to make this possible. The user interacts with the system using the neuGrid store, to search for and to identify an appropriately large set of images from a group of hospitals that match the required criteria. At this stage access controls and ethical policies are fully enforced to protect sensitive data. The researcher then uses the system to submit the study set for analysis through a workflow.

Once the data has been imported into the neuGrid system and users are able to access and query these data, they may like to carry out studies and carry out data analyses to find results of interest. Workflow development is a methodology that can be used to represent user preferences for an automatic analysis of data and this is the third stage in the analysis chain. Users may create workflows and then quickly execute the workflows on distributed resources provided by the Grid. The workflow development and execution is an important stage in the analysis life cycle in the neuGrid project. For example, a researcher may wish to run a comparative analysis using a study set of 3000 MRI scans stored in geographically distributed medical centres. It is important that the results are generated in a timely fashion as the researcher may have a number of different studies to carry out that week. The user interacts with the system to choose a study set of 3000 images, selects the pipeline or workflow through which the analysis will take place and starts the analytical process. Users can not only use the workflows and study samples that have been developed previously, they can also construct new workflows. For example, a new image analysis methodology may be developed and a researcher may wish to build a workflow to run it. Using an interactive creation tool the user can construct a workflow and specify some initial settings. The user may also create a record which describes the workflow and gives other users information about its purpose and access controls. The system allows different versions of the workflow to be created, tested and released when they are ready for use by other researchers.

It is not enough only to create workflows and execute them. It is important that results, as and when required, should be reproduced and reconstructed using the past information. The verification of the results using the audit trail information is known as provenance. The validation of results using provenance data is an important requirement in the analysis process. For example, a workflow yields some surprising and possibly significant results. A researcher may wish to confirm that the results are accurate and identify any mistake that may have been made. By analysing all the intermediary image sets and workflow execution logs the user is able to manually verify that the results were incorrect. It may be found that the error was due to a specific group of images interacting badly within the workflow, the user can then annotate the workflow so that other users are warned if they attempt a similar analysis.

Sometimes it may not be sufficient only to reproduce the results. It may also be necessary to validate and, if required, reproduce the workflow that has been used to obtain the results. This makes users confident not only on the results that have been produced but also on the process

that led them to generate these results. For example, a user may create a new workflow and run it on a test data set. At each stage in the execution of the workflow, the intermediary images or data are stored and a full provenance track is kept. After results have been produced, the user can examine the provenance to check that each stage of the analysis was completed correctly. The raw results can then be exported into the user's preferred analysis tool and the whole process can be added to the researcher's history for future reference. Initially the new workflow may produce some poor results during testing. The researcher therefore can inspect the logs of the workflow execution and locate the problem. The user can then interact with the system to make changes to the relevant settings and re-run the test study. This time the process may run correctly and meaningful results may be produced. Without the mechanism to validate workflows, it would not have been possible to correct the process and generate accurate results. Therefore validations of results as well as the workflows are two important requirements that should be addressed in the neuGrid system.

Once a workflow has been developed and verified, a user should be able to share it with other researchers in the field. The user should be able to make the workflow available to a team or group of users from a partner institution or project. This will save time, effort and resources from other teams and they will not have to reinvent the pipelines which have been produced by their peers or partners. Not only this, the user should also be able to share results and histories of their analysis processes. For example, a user may interact with the system to search existing studies and to compare, contrast and validate their results against research from other groups. This process helps the researcher to identify an error in their methodology and prevents them from making any embarrassing claims. The researcher may have carried out a similar study six months ago and may be worried that it too, might have been influenced by a similar error. The user can look up their research history and identify the appropriate study. The original process can be re-run on the original data set using the stored settings and pipeline configuration. This allows the researcher to confirm that the previous results were correct.

5. The neuGrid Services Architecture

Once the project requirements have been discussed, it is necessary to map the requirements against the components that can provide these features. As discussed in the design philosophy section, the components should act like services. The reasons for building these components as services have been presented in detail in the design philosophy section. Each of the stages in the end to end diagram (see figure 7) will be mapped to a suitable service that should have the capability to deliver the required functionality. These services will be autonomous and loosely coupled entities that can exist independently, can address requirements that cannot be addressed by any other component or service, and in cooperation with other services, can support the user analysis process. A more detailed and technical architecture of the higher-level end to end diagram discussed in the section 3 is presented below. As shown in figure 8, the overall architecture is divided into services. These services will address the project requirements as have been discussed in section 3 and will coordinate with each other to deliver a fully functional neuGrid system. In figure 1, the services are divided into three groups: 1) User facing services 2) Generic distributed services and 3) Grid facing services. A description of each group of these services is discussed in the following paragraphs.

The user facing services include the services which are accessed by a user for his or her day-to-day to activities. The user facing services coordinate with the generic services to provide the functionality which may be required by users to carry out their analysis. The user facing services are an entry point for the user to access the rest of the system and hide the underlying details and complexity of the overall system from a user. The first of these services is a *portal service*. The user will interact with the system through the portal service. The portal service

will have two components, one component provides the front-end interfaces and the second component links the back-end services (such as the generic services in WP6) to the front-end. A user can access most of the WP6 services through the portal service. The portal service has domain specific features as well as generic features. In WP6, only the generic aspects of the portal service will be designed and implemented. The domain specific aspects of the portal service will be addressed in coordination with WP5.

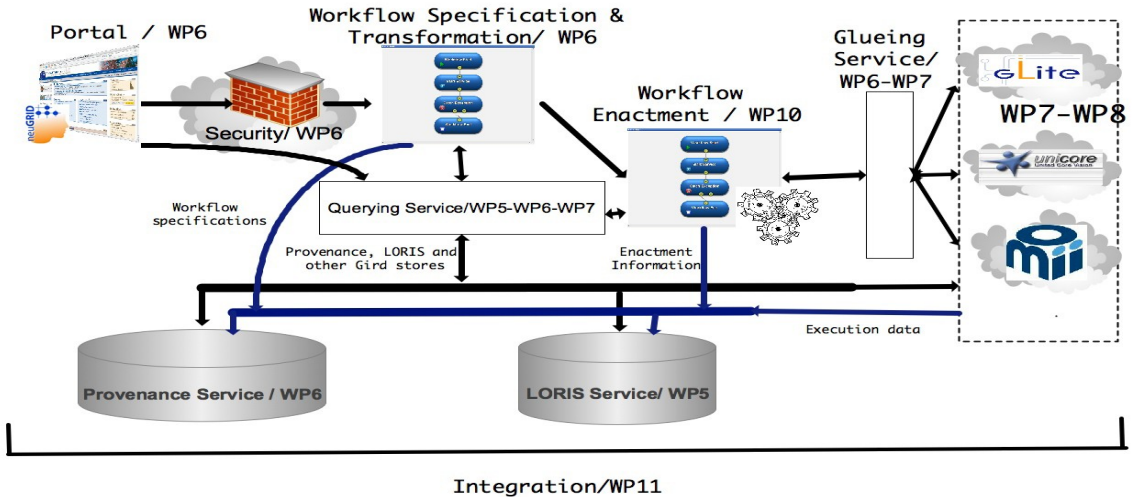

Figure 8: neuGrid Services Architecture

When a user tries to access back-end services through the portal service some form of authentication is necessary to ensure security and this is handled by the *security service*. This service will be responsible for all the authentication, authorization, access controls and policy enforcement issues. The security service, in association with the *anonymization service,* will also be responsible for the anonymization and privacy protection of the datasets which will be studied in an analysis. The anonymization service will also address the format conversion (for example from MINC to DICOM) issues. The anonymization service will be designed and implemented through the collaboration of WP6 and WP5. Similarly, the security service will be designed and implemented by WP5 (user facing services) and WP7 (Grid services) since front end and Grid facing services have major stakes in this service. Another important service that is grouped with the user facing services is the *LORIS service.* The LORIS service will be responsible for the data acquisition, capturing, annotating, visualizing, and tracking the acquired datasets and sharing it with other users and services. It will do the quality control and validation on the acquired images and will apply the necessary format changes to make them useful for the community. It will also be responsible for the schema creation, population and retrieval of the datasets from the LORIS database. Figure 9 describes the division of work between different work packages and how the services are grouped into layers is also depicted.

The second group of services mentioned in the services architecture (see figure 8) are the Generic Distributed Services. These services, as the name suggests, are not tied down to a particular application or Grid middleware. These are generic services, can be used in any application domain and can be deployed on any Grid infrastructure. They will be designed in such a way that a variety of applications and Grid middleware could be supported. This abstraction will be achieved through the use of high-level interfaces provided by different SAGA implementations and the *glueing Service*. The first of the generic services is the *workflow specification and transformation service*. The users, using this service, can specify their pipelines and workflows. This service will support all major authoring environments (such as Kepler, LONI, Taverna etc) and will transform the authored workflows into a common format using a translation component. These transformed workflows could

then be enacted in any execution environment. A user will also be able to download a specific workflow to his workspace and edit it by modifying algorithms in the workflow or changing input data sources or other settings. These changes will be done through a workflow-authoring environment, which will be integrated with the portal service. From within the portal service, the user can then submit the workflow for enactment and can view the results.

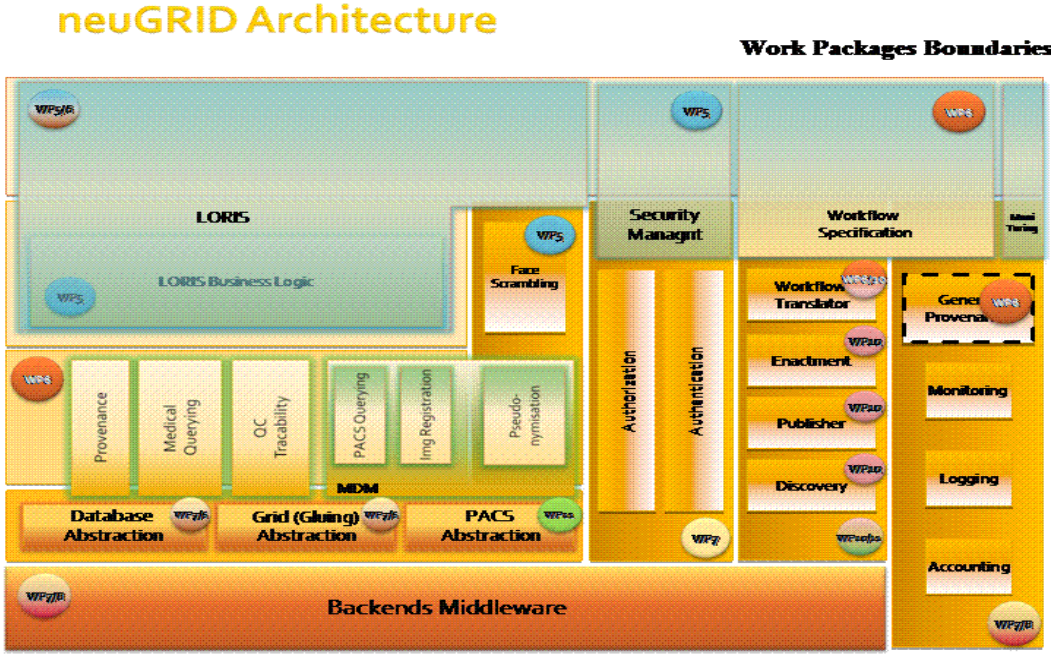

Figure 9: neuGrid Layered Architecture

Users may browse for available workflows through the *querying service*. The querying service will be a generic service that can also query and browse data that is stored in a file or relational database or in any other Grid database. The querying service will provide high level querying functionality for the Provenance store, LORIS store and other Grid data stores as may be required in future. In order to access and query data through the querying service, the user has to go through a security service, which may consist of either simple user/password or a public key infrastructure system. A short description of the security service has already been provided in this section. The portal service will provide a rich interface, which will allow users to perform numerous operations such as accessing workflows, which are versioned and maintained in the provenance store. Users can also view results from previous job executions, can access the output data from the LORIS store and can download this data into their workspaces. The *provenance service* as shown in the services architecture diagram is a generic service which will capture, store and perform an intelligent analysis on the data for better decision making. Provenance is the process of tracking the origins of data and its evolution between different stages and services. The provenance service will provide the means for capturing and maintaining workflow specification and execution information in a provenance database. Users will be able to query provenance information in order to observe the behaviour of workflow specifications and past executions. An intelligent feedback system, built from the provenance information, will enable a user to specify workflow components for their optimized execution.

Once a workflow has been specified and associated data elements have been identified through a workflow authoring environment and the specified workflow have been transformed into an executable format, the workflow is ready for enactment. This enactment process is managed through an *enactment service*. The enactment service takes a workflow

specification as an input, queries the data that is required to run the workflow, parallelizes the workflows according to some performance and efficiency considerations and dispatches the workflows for execution to a Grid environment. The enactment service has two components. One is the logical part where all the performance and parallelization decisions are taken and this is quite generic. The other is the core enactment part where the workflow is prepared for execution and that may be middleware specific. Therefore, the former part will be addressed in WP6 that is responsible for generic services and the later component will be designed and implemented in WP10.

Before the workflow is actually passed over to a Grid environment, it is handed over to the glueing service, as shown in the services architecture diagram, which has a major role to play before execution. The workflow enactment carries out the execution of the workflow through the Glueing service on the Grid. The Glueing service hides the encapsulation of Grid middleware complexities from the neuGrid services. Using the glueing service, the workflows, queries and other jobs can be submitted to any Grid environment that could be built upon gLite, Globus, Unicore or any other Grid middleware. In essence, the glueing service is the gateway which shields the services from being locked in to a particular Grid middleware and makes them truly generic. Once the glueing service passes over the workflow to a Grid scheduler, it is managed by a particular Grid middleware and scheduled on the distributed resources for execution. The distributed infrastructure in the neuGrid project will comprise Grid services that could include scheduling and workload management services, replica location and replica selection services, virtual organization membership service, computing and storage services and many other Grid services that may or may not be a part of a particular Grid deployment. When a job or workflow is being executed in a Grid execution environment, its progress and corresponding input and output operations are being monitored and logged into the provenance database. The results of the workflow execution are stored both in the LORIS and provenance data stores and are handled through their respective services.

6. An Overview of the Distributed Medical Services

Figure 9 outlined the roles of the work packages in the neuGrid project and how the services layers could fit into these work packages. Three groups of services were identified. The first group of services was identified to be very domain specific, containing the neuGrid business logic. These will be the user facing services and a user will mostly access these services to author workflows and visualize the results. A second group of services was identified that will always sit on the Grid middleware and will provide the necessary infrastructure to execute these workflows. These Grid facing services are very middleware specific and always depend on the particular middleware being used to provide the distributed Grid infrastructure. The third group of services that was identified is neither middleware- nor application-specific. These generic services can be used by any application and can run on any Grid middleware. It is these generic distributed services are the services that will be covered in WP6. In the following sections, we briefly outline each of the generic services. Major components and the role of the services are identified within the services architecture. The detailed design of the services is presented in the respective Service Design documents.

6.1 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.

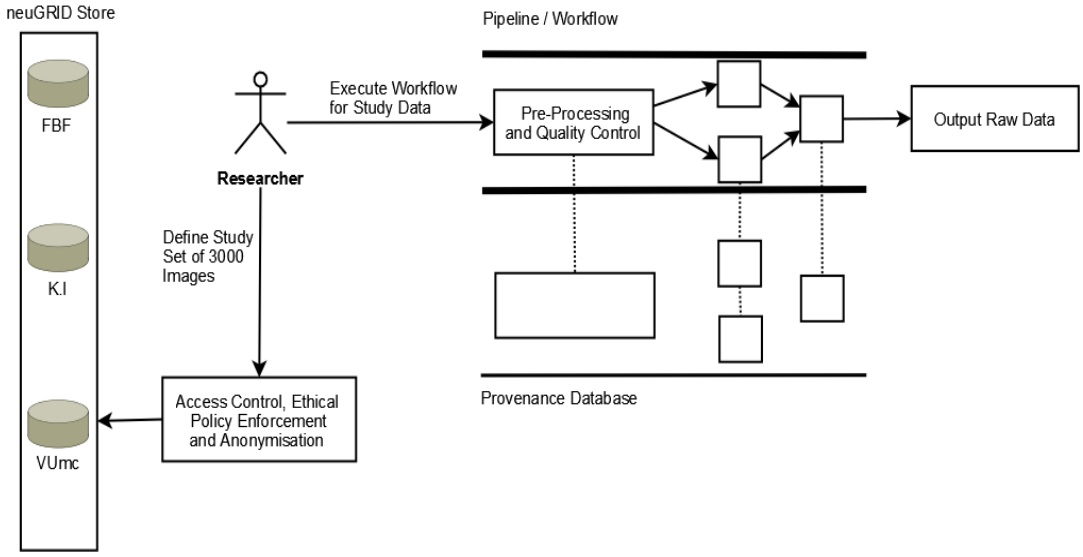

Figure 10: A conceptual model of a neuro-imaging Pipeline

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.

Relevant neuGrid user requirements and the WP6 Services Design Philosophy guided the design of the Pipeline Service. 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. The Pipeline Service design document also outlines future research issues. 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.

6.2 The Glueing Service

The medical domain is increasingly building high-level distributed services such as querying, workflow creation, scheduling services, image handling and repository services. Such services help medical users to analyse their data, extract knowledge from it and share the results with other users. Most of these services are designed and developed for a particular community of medical users. The services built by one community often cannot be shared or re-used in other medical domains due to architecture, interface or platform limitations. The Glueing Service addresses the aforementioned issues by proposing an architecture which shields the heterogeneity of distributed resources and enables a set of generic services to access Grid resources without getting into the details of underlying Grid middleware or architecture. The Glueing Service is a constituent service of the generic middleware services layer in neuGrid, which aims to provide:

- A standard way of accessing Grid services without tying services and applications to a particular Grid middleware
- A mechanism to access any deployed Grid middleware through an easy-to-use service.
- A solution which extends and enhances the reusability of already developed services across domains and applications
- A service-based approach to shield users and applications from writing complex Grid specific functionality. The user requires a minimum set of Grid specific APIs and the rest of the functionalities are managed by the service.
- A simplified approach for enabling clients to gridify their applications without installing and maintaining too many Grid specific libraries.

The Glueing service will expose the SAGA [15] (Simple API for Grid Applications) implementation using a web service to meet the aims mentioned previously. SAGA is an open standard that is defined and maintained by the Open Grid Forum [16] (OGF), which describes a high-level interface for effortless programming of Grid applications. SAGA is not designed for middleware developers, rather it provides APIs for developing applications using different Grid middleware. The SAGA API allows running a job on a particular middleware by the runtime loading of a middleware adaptor. SAGA middleware adaptors pass jobs to the middleware as its clients and are responsible for low-level communication with it. This shields the low-level middleware difficulties from the user and will encourage him to use them with little or no knowledge.

The Glueing service design addresses some major requirements of the neuGrid project and will provide a generic framework for accessing resources over the Grid. The heterogeneity of distributed resources and details of grid middleware architectures will be transparent from users. The design of the Glueing service is based on SOA principles, which will help different services in neuGrid to use service functionalities through standardized interfaces. This will

also allow other client applications to use service features by inspecting its WSDL, available online on the service endpoint URL. SOA-based architecture of the Glueing service is in line with the project requirements and will provide a gateway to all WP6 service to access grid resources.

6.3 The Provenance Service

The process of neuroimaging analysis in neuGrid will involve a number of steps, an execution failure at any stage may lead to undesired execution results. These may be caused by incorrect pipeline specifications, inappropriate links between pipeline components, execution failures because of the dynamic nature of the Grid and others. A real problem in this scenario is tracking faults as and when they happen. This is mainly because of the absence of an information capturing mechanism during the pipeline specification, gridification and execution. Thus a user is unable to track errors in past neuroimaging analyses. This may lead to a loss of user control or repetition of errors during subsequent analyses. Users may face a range of problems in such cases, which may prevent them from being able to:

- Reconstruct a past pipeline or parts of it to view the errors at the time of specification.
- Validate a pipeline against a reference specification.
- Validate pipeline execution results against a reference dataset.
- Query information of his interest from the past analysis.
- Compare different analyses.
- Search annotations associated with a pipeline or its components for future reference.

To address the aforementioned problems, the generic medical services layer of neuGrid involves a process of keeping track of the origins of the data and its evolution between different stages and services. This process is called provenance and it will allow users to query analysis information, automatically generate analysis pipelines, detect errors and unusual behaviours in past analyses, and validate analyses. Tracking provenance data is important as it is useful in identifying the problems and errors that commonly occur during neuroimaging analysis. Provenance will support the continuous fine-tuning and refinement of the pipelines by capturing:

- Pipeline specifications.
- Data or inputs supplied to each pipeline component.
- Annotations added to the pipeline and individual pipeline component.
- Links and dependencies between pipeline components.
- Execution errors generated during analysis.
- Output produced by the pipeline and each pipeline component.

The neuGrid infrastructure will incorporate a provenance service, which will interact with other medical services through standardized interfaces. This service-oriented approach also allows centralized management and the on-line availability of the provenance service. A conceptual model of the provenance service and a scenario of user interaction is shown in the Figure 11. The next few sections of this document explain the user requirements of the provenance service, the design, service components and relevant technologies that have been evaluated in the light of user requirements.

Provenance service will provide recording and querying interfaces to store pipeline information and retrieve already stored provenance information. These interfaces will be published to user communities, following SOA design principles. Therefore a set of APIs will

be exposed, making the service functionality available online. The provenance service APIs will also allow a user to retrieve a complete workflow or parts of it from the provenance database. This information will be viewable in the workflow authoring environment, allowing a user to reconstruct a past analysis.

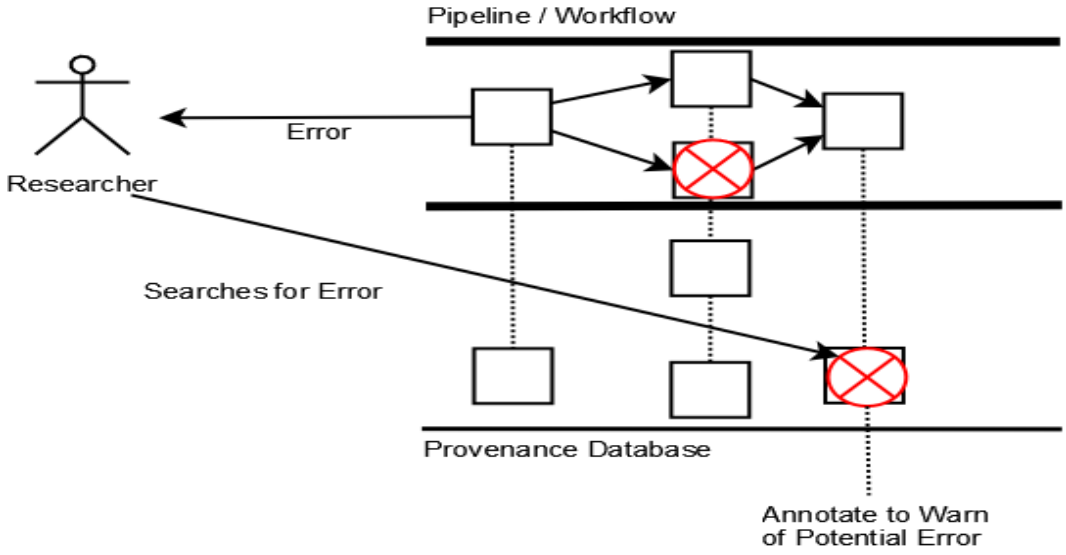

Figure 11: A conceptual model of the Provenance Service

This process will help in validating an analysis against a reference dataset and re-execute a pipeline or parts of it. The provenance service will also facilitate other WP6 services because of its SOA-based design. This is also inline with the neuGrid project requirements, which require all services to communicate with each other through their standardized interfaces. Besides aforementioned service features, workflow provenance has many potential research aspects. One such research area is the classification of provenance data in different categories and modelling provenance information using WF specification and execution details. Moreover finding the best specification and execution models and providing user a feedback with such information will enable him to specify workflows for their optimized execution. These research areas will be explored if time and resources permit after the completion of provenance service.

6.4 The Querying Service

The neuGrid project is centred around the provision of a research infrastructure for the analysis of complex medical data. Heterogeneous sources of relatively complex data are clearly present within clinical research studies and are therefore difficult to integrate. Integration is needed to ensure that researchers get the most relevant and subsequently, the best information for their research. The querying service will provide methods by which heterogeneous data can most efficiently be queried within neuGrid and hence assist users in their research. The primary emphasis, being on allowing users to query the data successfully, in the future this could progress into developing ways to assist the user i.e. adding semantics to give the querying service a level of intelligence. The data, despite being heterogeneous in nature could also be in many different formats. Examples of these include images, flat files, relational databases and XML to name just a few that the querying platform should be flexible enough to handle. This service will provide a choice of ways in which the user can query the data held in neuGrid.

The primary issue with this service is combining the heterogeneous sources of data together so that they can be queried as a single resource. Once this has been accomplished, work will

be focused into including some semantics in order to enrich the querying offered to researchers to browse and access the distributed data resources. The fundamental way in which semantics are currently applied to aid existing systems' intelligence will be analyzed in the evaluation process. Current thoughts suggest that semantics will be employed to provide a very flexible query service offering lots of types of query mechanisms for the users.

The neuGrid project design philosophy identifies service-oriented architecture as being the best choice for addressing the user requirements. It is clear that querying will play a central role in the system by providing mechanisms for accessing the distributed data resources, which are inside the grid environment. This service will research and develop a platform independent and developer friendly querying service, which can be customised and extended in the future. Local data can often be specific to individual institutions and therefore structured in different ways. Such data often comes in a variety of formats and it is challenging to query and integrate it. Rather than having a specific data-adaptor approach for each type of resource, it would be beneficial to use a service-oriented approach to provide the level of abstraction and scalability that is necessary.

Ultimately, the goal is to create a synergy between the distributed querying service and semantic information. This is relevant to the service because in some cases researchers will type a query which in itself will return very specific results from heterogeneous data resources. The aim is to bridge gaps between distributed data resources using information contained within the querying service itself and to make inferences as the software is running thus adding to the services own knowledge base. It is this very bridging of data within separate data resources, which could be of interest because that is something not easily done by a human with limited time.

It is important to remember the possible impact of such a service being implemented, it is reasoned that researchers may be less explicit in their searches and still receive relevant search results. This service should help to close the gap between distributed querying and semantic integration in order to provide richer results

The following objectives of the querying service have been identified:

- Create a querying service, which can query disparate data resources.
- Craft a solution that is platform independent and service oriented.
- Create a synergy between the querying of heterogeneous data resources and semantics, something which based on initial research, there are very few systems in existence accomplishing this.
- Look at semantics fundamentals and how they can be applied to aid the querying service and bridge gaps between data resources within the service to yield more useful results.

The design of the Querying Service has been guided by the user requirements and a web service will be implemented to allow flexible access to neuGrid data. The proposed interface has been discussed along with the communication down through the layers to the underlying data resources. It became apparent early on that adopting a web service alone lacked the semantic capabilities, which would be required to make the service as flexible as is desired. Methods of enrichment via semantics have been discussed along with the need for the presence of such features within the querying service. It is thought that these methods of semantic enrichment will keep the querying service as flexible as possible whilst providing the user with an array of querying options.

An API will be developed forming a standard interface to the querying service. This will make the service as accessible to human users as it will be to other services. The vision being that, a user provides a query and the service cooperates with a rule engine and an ontology to determine what the users intentions are and if there are possibilities for query enrichment.

6.5 The Portal Service

The Portal Service will be the single point of entry for users to access the neuGrid services. It will hide the complexity of the underlying low-level neuGrid architecture from the users and will enable them to focus on using the services' functionality. It will allow the community users to easily authenticate, access the services, browse the data, launch analysis and visualize their results. The Portal Service is designed to provide access to other services in the neuGrid infrastructure using simple interfaces. It will integrate all of them into one single User Interface, hiding all the architecture internals and calling individual Web Services according to the needs of the users and the application workflow.

The Portal Service requirements (which have been collected in work package 9) state the following important functionality to be addressed in the portal service:

- It should have an easy to use interface
- It should have a flexible interface that could be easily customized and reused
- It should be the single point of contact to access the underlying services that may be further composed with the different low level services

The Portal Service will help to:

- Authenticate users
- Access services
- Browse data: access and query data and intermediate/final results
- Launch analyses
- Visualize the produced results
- Track and manage workflows
- Submit Jobs and carry out related steering and monitoring
- Manage users/groups/rights/security through respective services and well-defined ACLs
- Manage services (registration and removal)
- Allow access to both Grid and non-Grid services alike

Due to the very specialized and original architecture of the neuGrid system, the usage of a Grid-aware portal would probably only add constraints making it difficult to implement the Service Oriented Architecture that has been identified by the consortium as a means of constructing the infrastructure. Since Grid-aware portals are used to integrate all components into their own specific architecture, an implementation of this kind of framework in neuGrid would not be possible without sticking to designs which are not as flexible as the one required in neuGrid, particularly in terms of technical abstractions and decoupling. Therefore it seems reasonable to consider more general portal environments and there are a wide range of such technologies. Clearly the widely used AJAX platform has a range of features that makes it well suited to integration with web-services. This has led to its adoption as a candidate technology for providing the user facing aspects of neuGrid.

From the desktop application standpoint, the need to reuse what already exists (namely specialized and well designed Web interfaces) and the requirement to create an easily upgradable architecture are points, which certainly require further evaluation. So, from all these current evaluations and analysis it seems that the most flexible and simple approach and therefore the most reliable is the federation of all the web applications using a Single Sign On facility. This will provide the users with a comprehensive and harmonized interface. The creation of a dedicated portal aggregating service, without an overly restrictive web interface, will enable users to add the missing building blocks to the portal. This will allow neuGrid to offer an harmonized and federated set of specialized and dedicated interfaces to the users. It is

reasoned that this addresses the requirements of users and will deliver a satisfactory user experience, thereby encouraging the wider adoption of the neuGrid platform.

6.6 The Anonymization Service

The neuGrid platform is intended to handle a large quantity of sensitive data coming from heterogeneous sources. It is extremely important therefore, to ensure that medical information is not made available without appropriate ethical clearance. Such legal and ethical requirements mean that an efficient and common level of anonymization must be used throughout the project. Anonymization should therefore be considered at the following two levels:

- Pseudonymization: This is defined by Wikipedia as "a procedure by which all person-related data within a data record is replaced by one artificial identifier (like a hash value) that maps one-to-one to the person. The artificial pseudonym always allows tracking back of data to its origins which is the difference with anonymized data, where all person-related data that could allow backtracking has been purged." [16]

- Face scrambling: This is the process by which algorithms or manual processes are applied so that the face is removed from an MRI image, thus preventing the possibility of a subject being recognised.

In the context of neuGrid, anonymization is the means by which it is ensured that data cannot be traced back to the originating subject. This is a challenging task and is informed by the work of WP2 which is actively considering the level and type of anonymization that should be applied. Given the legal complexities that surround this subject, the design of the anonymization service has necessarily progressed at a somewhat slower pace than some of the other services. This is to be welcomed because it is essential that care be taken in complying with national and international policy. It is clear that for neuGrid to become a successful research infrastructure, such issues will need to be addressed thoughtfully and with care. It is with this purpose in mind that an initial outline is presented of what an anonymization service may contain. This is seen as an important step in stimulating discussion and driving a well founded design that will be refined and covered in more detail in subsequent project deliverables.

The purpose of the anonymization service is to facilitate the pseudonymization of the data that is stored within the neuGrid infrastructure in order to make it available to users, so that they can use it in their analyses whilst preserving the anonymity of the patients. The pseudo-anonymization process will include the checking of files to ensure that all the markers, which can provide information to identify the patient, are removed before the image can be made available. Most of the time it will be done by removing the image file's text headers containing metadata about the patient such as name, date of birth or any other information that could identify them. The face scrambling process will try to make sure that faces on images cannot be recognized. The anonymization service interacts mainly with the PACS Abstraction, the Grid Abstraction and the Face Scrambling services.

Privacy of patients is one of the key requirements that the neuGrid project must address in order to be positively accepted and widely used by medical communities, which could benefit from the neuGrid infrastructure. If users can utilise the architecture to widely share their patients' data in confidence, with care taken on the privacy of their data, this will greatly assist the adoption and the usage of such an architecture. By providing different steps of pseudonymization and face scrambling, taking place at different levels of the data acquisition process, a high level of privacy protection can be achieved.

As highlighted by the neuGrid protocol for Data Protection document (**D2.3**) all the possible ethical and legal problems related to the patient privacy have been resolved, both by following the HIPPA's recommendations during the writing of the appropriate anonymization architecture and by requiring the informed consent of the subject. Finally, this architecture is as close as possible to the implementation of what the consortium agreed upon at the CERN meeting on December 2008 where all possible architecture models were discussed. This architecture will hopefully address the requirements of the project in the best possible manner.

7. Conclusions and Future Work Plan

The first year of the project concentrated on eliciting the user requirements and mapping user requirements to specific services. The services design philosophy was defined in order to make the neuGrid services scalable, re-usable, extensible and compliant to generally agreed standards. The services have also been designed to be middleware agnostic and loosely coupled. The service designs have been guided by the user requirements and the WP6 design philosophy. The designs have been created after extensive evaluations of relevant state of the art technologies. More detailed designs of each generic service will follow the issue of this design strategy document and will be updated as and when needed (e.g. at project milestones) through the neuGRID project.

The second year activities include the following:

1. Detailed design specifications and definitions of the APIs of the individual services will be created. The API specifications will define functionality each service will provide. The services designs and their API specifications will facilitate in creating more interoperable services infrastructure.

2. Detailed plans for implementing the services according to the design philosophy and user requirements will be created. The functionalities will be identified and prioritised in each of the services that are really required to address the user requirements.

3. The design specification produced and the architecture proposed will be discussed with the potential user communities. The intended end-users will verify the functionality of each of the services, choice of the technologies that have been made to address the requirements and immediate outcome of the proposed implementation plans.

4. After end-user agreement, initial prototype of the services will be developed. These prototypes will have the basic functionality that is mandatory to address the essential user requirements.

5. The user and technical guides and an implementation report for each of the services will be compiled and presented to the users for their feedback and evaluations. A quality assessment mechanism will also be put in place to ensure that services are designed and delivered as per the design philosophy and meet the user expectations.

6. The prototype implementations will be designed to be feature complete, however they will not be the final deliverable services. The end users will evaluate prototype services, and their feedback will be used to generate more accurate service design specifications.

7. The feedback gained from the end-users will be used to create more enhanced services which will be the focus of future activities in the project. The future activities will also include steps to optimize the service implementations and their

benchmarking specifications will also be produced. Service standardization measures will also be introduced to make them interoperable and widely exploitable across communities and application domains.

8. References

[1] M. Kuba, 2005. Integration of medical services using semantic grid techniques. Technol. Health Care 13, 5 (Dec. 2005), 410-411.

[2] F. Estrella et al, A Service-Based Approach for Managing Mammography Data, Submitted to the 11th World Congress on Medical Informatics (MedInfo'04) San Francisco, CA, USA. September 2004.

[3] DICOM Digital Imaging and Communications in Medicine, http://medical.nema.org

[4] EU Share Project, http://eu-share.org/about-share.html

[5] The Open Group, Service Oriented Architecture, http://www.opengroup.org/projects/soa/doc.tpl?gdid=10632

[6] Service Oriented Architecture, Wikipedia,http://en.wikipedia.org/wiki/Service-oriented_architecture

[7] Joe McKendrick, SOA's loose coupling makes performance difficult to monitor, March 21st, 2006, http://blogs.zdnet.com/service-oriented/?p=576

[8] Baskerville, Richard et al, Extensible Architectures: The strategic Value of Service Oriented Architecture in Banking, European Conference on Information Systems (ECIS), 2008

[9] Interface Description Languages, http://en.wikipedia.org/wiki/Interface_description_language

[10] Web Services Enumeration (WS-Enumeration), W3C Member Submission 15 March 2006, http://www.w3.org/Submission/WS-Enumeration/

[11] John J. Barton, Satish Thatte, Microsoft,Henrik Frystyk Nielsen, SOAP Messages with Attachments, W3C Note 11 December 2000,http://www.w3.org/TR/SOAP-attachments

[12] Steve Graham et al. Web Services Notification,March 2004, http://download.boulder.ibm.com/ibmdl/pub/software/dw/library/ws-notification/WS-BaseN.pdf

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

[14] http://saga.cct.lsu.edu

[15] http:/www.ogf.org/documents/GFD.90.pdf

[16] Pseudonymization http://en.wikipedia.org/wiki/Pseudonymization