EGI-InSPIRE

EGI Software Repository Architecture and Plans

EU MILESTONE: MS504

Document identifier:	EGI-MS504-final
Date:	**05/11/2010**
Activity:	**WP5**
Lead Partner:	**GRNET**
Document Status:	**FINAL**
Dissemination Level:	**PUBLIC**
Document Link:	https://documents.egi.eu/document/89

Abstract

This document describes the current design and architecture of the EGI Software Repository (http://repository.egi.eu) and associated support tools (http://www.egi.eu). It also discusses the future plans for year 1 of the EGI-InSPIRE Project.



I. COPYRIGHT NOTICE

II. DELIVERY SLIP

	Name	Partner/Activity	Date
From	Kostas Koumantaros	GRNET/WP5	03/08/10
Reviewed by	**Moderator:** Daniele Cesini **Reviewers:** Karolis Eigelis	SA2 EGI.eu	04/08/10
Approved by	**AMB & PMB**		28/09/2010

III. DOCUMENT LOG

Issue	Date	Comment	Author/Partner
0.1	13/07/10	First draft	K. Koumantaros/GRNET W. V. Karageorgos/IASA
0.2	28/07/10	Second draft	W. V. Karageorgos/IASA S. Sotiropoulos/IASA M. Chatziangelou /IASA
0.3-0.14	2/8/10	Third Draft	K. Koumantaros/GRNET M. Liška/CESNET C.Theodosiou/AUTH
0.15	3/8/10	Fourth Draft	K. Koumantaros/GRNET
0.16	3/8/10	Final Draft	S. Newhouse/EGI K. Koumantaros/GRNET M. Chatziangelou /IASA
1.0	28/09/10	Addressing the Comments from the Reviewers.	K.Koumantaros/GRNET

IV. APPLICATION AREA

This document is a formal deliverable for the European Commission, applicable to all members of the EGI-InSPIRE project, beneficiaries and Joint Research Unit members, as well as its collaborating projects.

V. DOCUMENT AMENDMENT PROCEDURE

Amendments, comments and suggestions should be sent to the authors. The procedures documented in the EGI-InSPIRE “Document Management Procedure” will be followed: https://wiki.egi.eu/wiki/Procedures

VI. TERMINOLOGY

A complete project glossary is provided at the following page: http://www.egi.eu/results/glossary/.

VII. PROJECT SUMMARY

To support science and innovation, a lasting operational model for e-Science is needed – both for coordinating the infrastructure and for delivering integrated services that cross national borders.

The EGI-InSPIRE project will support the transition from a project-based system to a sustainable pan-European e-Infrastructure, by supporting 'grids' of high-performance computing (HPC) and high-throughput computing (HTC) resources. EGI-InSPIRE will also be ideally placed to integrate new Distributed Computing Infrastructures (DCIs) such as clouds, supercomputing networks and desktop grids, to benefit user communities within the European Research Area.

EGI-InSPIRE will collect user requirements and provide support for the current and potential new user communities, for example within the ESFRI projects. Additional support will also be given to the current heavy users of the infrastructure, such as high energy physics, computational chemistry and life sciences, as they move their critical services and tools from a centralised support model to one driven by their own individual communities.

The objectives of the project are:

1. The continued operation and expansion of today's production infrastructure by transitioning to a governance model and operational infrastructure that can be increasingly sustained outside of specific project funding.
2. The continued support of researchers within Europe and their international collaborators that are using the current production infrastructure.
3. The support for current heavy users of the infrastructure in earth science, astronomy and astrophysics, fusion, computational chemistry and materials science technology, life sciences and high energy physics as they move to sustainable support models for their own communities.
4. Interfaces that expand access to new user communities including new potential heavy users of the infrastructure from the ESFRI projects.
5. Mechanisms to integrate existing infrastructure providers in Europe and around the world into the production infrastructure, so as to provide transparent access to all authorised users.
6. Establish processes and procedures to allow the integration of new DCI technologies (e.g. clouds, volunteer desktop grids) and heterogeneous resources (e.g. HTC and HPC) into a seamless production infrastructure as they mature and demonstrate value to the EGI community.

The EGI community is a federation of independent national and community resource providers, whose resources support specific research communities and international collaborators both within Europe and worldwide. EGI.eu, coordinator of EGI-InSPIRE, brings together partner institutions established within the community to provide a set of essential human and technical services that enable secure integrated access to distributed resources on behalf of the community.

The production infrastructure supports Virtual Research Communities (VRCs) – structured international user communities – that are grouped into specific research domains. VRCs are formally represented within EGI at both a technical and strategic level.

VIII. EXECUTIVE SUMMARY

This document describes the current architectural design of the EGI Software Repository (http://repository.egi.eu) and associated support tools (http://www.egi.eu). It also discusses the future plans for year 1 of the EGI- InSPIRE Project. Section **Error! Reference source not found. Error! Reference source not found.** covers the requirements defining the architecture and functionality that the EGI Software repository and associated support tools need to provide. It briefly discusses the relevant use cases in an effort to identify all the relevant actors and the corresponding workflow needed for the EGI Software Release cycle. Section 3 Architecture describes the current implementation plan of both the Resource Tracker that implements most of EGI Software Release cycle workflow and the EGI software Repository 3 tier Systems architecture including the interface between them. As the requirements to the EGI Software Repository will be adapted to both the needs of the user community it is targeted to serve and the outcome of the negotiations with the technology providers, the plan for the next year foresees a number of iterative releases in M7 to M12 that will address any bugs or limitation of the system. These foreseen changes will be documented with yearly updates of this milestone that are scheduled for the duration of the EGI-InSPIRE project.

TABLE OF CONTENTS

1 INTRODUCTION

This document describes the current architecture of the EGI Software Repository (http://repository.egi.eu) and associated support tools (http://www.egi.eu) and the plans for the first year.

2 REQUIREMENTS CAPTURE

2.1 Introduction

In this section we present the requirements for a reliable and functional EGI Software Repository. We also try to establish a common vocabulary and schema to be used for the interaction with the software providers. Given that this document is heavily dependent on the Roadmap for the UMD software release, some parts carefully mirror the decisions outlined in it.

For the most part, the following sections outline the schema that will be used when information provided by the technology providers through the RT is transferred to the repository. The metadata that are provided to the RT are stored to the EGI Repository and used locally to provide meaningful semantics to the users. The proposed information schema (outlined in Section 4.3) is rich enough to allow all parts of the repository infrastructure (YUM/APT repositories and the web portal) to extract the necessary information.

2.2 Use case and user community

There are several kinds of people involved with EGI Software Repository, each of them holding different positions and roles. The main categories are the software providers, the evaluators/verifiers, the end users and the repository administrators. End users, in this context, are considered both grid site administrators and researchers/grid users of the EGI infrastructure. The software providers can be divided with respect to the kind of software that they provide. There is fully supported software (e.g. EMI), associated software and user community software. Depending on the kind of software, different roles of people are involved. For disambiguation purposes, please refer to the Glossary, provided as Appendix 1.

The fully supported software involves the most of them in the roll-out procedure. According to this procedure, the software provider makes sure that his/her packaged release/component meets all the agreed criteria [R 9] and inserts the necessary metadata information associated with the release. Based on the URL provided by the software provider, the release is downloaded to a preliminary area of the EGI Software Repository and after a set of checks the newly contributed software release is moved into the /scratch sub-repository. Then, the evaluators/verifiers get notified and the verification/reviewing process begins. If the release complies with the verification criteria, then is tagged as “Verified-InStageRollout” and is moved to the /stage-rollout repository and to the appropriate YUM/APT repositories. Afterwards, the evaluators get notified that a new release has just been moved/released and once more, the verification/reviewing process begins. If the release complies with the verification criteria, then the release is set as “Verified-InProduction” and is moved/released into the /production repository and to the appropriate YUM/APT repositories.

If the released package is an associated software, then the /stage-rollout step is missing and the software moves directly from the /scratch to the /production directory. Finally, the user community software does not pass any evaluation and is placed directly into the production repository. In these kind of packages evaluators/verifiers are not involved. The only people that are involved are software providers and end users.

The end users are intended to use the repository only for downloading the packages. This can be done through a YUM/APT repository or through the EGI Software Repository portal. Finally, administrators intervene to the repository outside the roll-out procedure. They perform administrative tasks such as managing YUM/APT repository subsystem or debugging errors on the package release procedure.

2.3 Software definition schema

The repository organizes information using the following granularity levels:

- **Package:** A package contains all necessary (binary, source or plain) files for a part of software. It might depend on further virtual or physical packages (see [R10]). Generally, a UMD package is no different from a standard distribution package of, for example, a regular Linux distribution. This is the smaller granularity level for the EGI Repository.
- **Software Component:** A software component provides certain grid functionalities, and at least from the point of view of the repository, it can be approximated by a Virtual Package and its dependencies.
- **Capability:** A capability represents a particular functionality necessary for the middleware stack, but in as much self-contained (i.e. not dependent on other functional elements).
- **Release:** A UMD Release is the largest element of the EGI Repository. It contains all software components and their associated packages, organized by capability, at a particular point in time.

The above definitions are not official, in the sense that they are meant to provide a more solid technical foundation than the definitions that are outlined in Appendix 1. Also, in the same appendix, a definition of the metadata elements is provided, so that the software providers can enter them appropriately and users can use them more efficiently.

Given that the use of capabilities is a shift from the previous software development and release paradigm (under EGEE), some of the metadata still reflect the previous status quo. This is desirable for the moment, as the interoperability between different software component families is a work in progress, and therefore for the immediate future, a degree of separation is necessary. As software development progresses, the metadata will become more meaningful.

In the following two sections, we describe two approaches for the XML file that defines the metadata necessary for the EGI Software Repository. As previously mentioned, the detailed definition of the metadata is provided in Appendix 1. The first approach assumes a flat structure, where only the smallest granularity level is taken into account when defining a release, while the second uses a structured XML file where the structure of a UMD Release - is provided as well. The pros and cons of each approach are described in the relevant section.

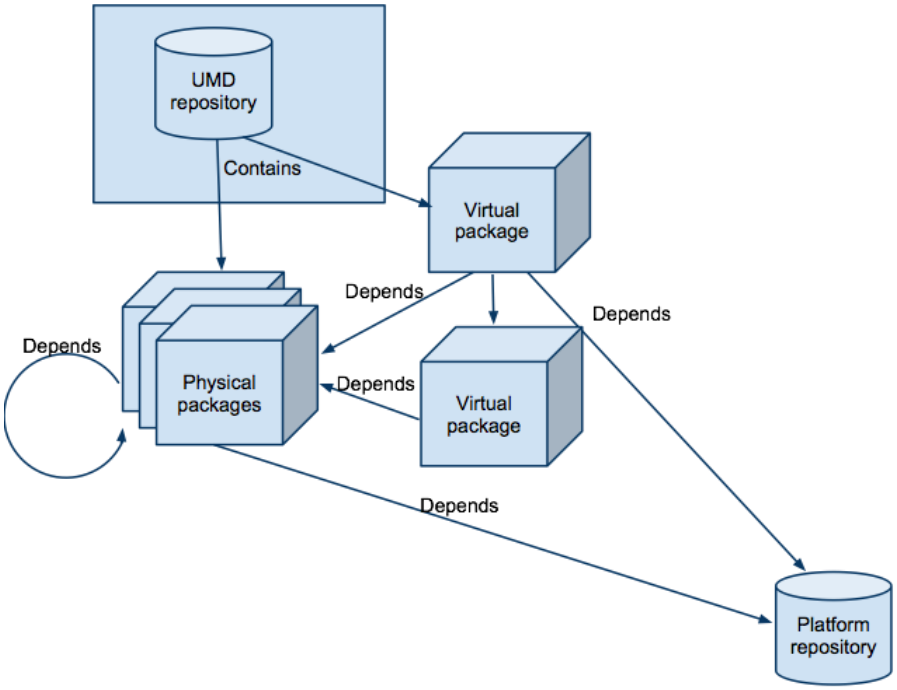

Figure 1: The relationship between virtual and physical packages within the EGI Software Repository.

2.3.1 Option 1: Flat XML file

In this case all updated packages are described sequentially, and all relevant release information is provided for all packages. Nestedness is determined by the UMD repository.

This makes updates relatively trivial, when only individual packages within a release are changed. It also allows the EGI Repository to decide whether the updates warrant a new UMD Release, or are considered trivial and therefore the UMD Release version should remain the same and the changes transparent to the users. However, recognizing a major structural change between releases (e.g. the addition or removal of a capability) is made more arduous. An example file follows, that defines two packages in a release, one virtual and one physical, with the virtual package depending on the physical.

Example:

```
<package Name="glite-WN-manpages" Type="Virtual" SLDType="community">
        <Capability>Computing</Capability>
        <SoftwareProvider>EMI</SoftwareProvider>
        <SoftwareComponent>glite3.2-WN</SoftwareComponent>
        <Creator>eangelou</Creator>
        <Contact>eangelou@cslab.ntua.gr</Contact>
        <Description>Provides glite-WN with manpages for the appropriate tools</Description>
        <License>GPL v.3</License>
        <Keywords>gLite gLite-WN man manpages</Keywords>
        <SupportedPlatforms>SL5</SupportedPlatforms>
```

```
        <Architecture>any</Architecture>
        <DocumentationLinks>http://www.cslab.ece.ntua.gr http://www.egi.eu<DocumentationLinks>
        <Date>21/6/2010</Date>
        <Version>3.2.1</Version>
        <ReleaseNotesURL>http://www.cslab.ece.ntua.gr</ReleaseNotesURL>
        <ChangeLogURL>http://www.cslab.ece.ntua.gr</ChangeLogURL>
        <RepositoryURL>http://repository.egi.eu/</RepositoryURL>
        <DocumentationURL>http://www.cslab.ece.ntua.gr</DocumentationURL>
        <Dependencies>glite-WN-manpages-3.2.15 manpages</Dependencies>

</package>

<package Name="glite-WN-manpages-3.2.15" Type="Physical" SLDType="community" Method="apt">
        <Capability>Computing</Capability>
        <SoftwareProvider>EMI</SoftwareProvider>
        <SoftwareComponent>glite3.2-WN</SoftwareComponent>
        <Creator>eangelou</Creator>
        <Contact>eangelou@cslab.ntua.gr</Contact>
        <Description>Provides glite-WN with manpages for the appropriate tools</Description>
        <SLDType>community</SLDType>
        <License>GPL v.3</License>
        <Keywords>gLite gLite-WN man manpages</Keywords>
        <SupportedPlatforms>SL5</SupportedPlatforms>
        <Architecture>any</Architecture>
        <DocumentationLinks>http://www.cslab.ece.ntua.gr http://www.egi.eu<DocumentationLinks>
        <Date>21/6/2010</Date>
        <Version>3.2.15</Version>
        <ReleaseNotesURL>http://www.cslab.ece.ntua.gr</ReleaseNotesURL>
        <ChangeLogURL>http://www.cslab.ece.ntua.gr</ChangeLogURL>
        <RsyncURL>ftp://www.cslab.ece.ntua.gr/</RsyncURL>
        <DocumentationURL>http://www.cslab.ece.ntua.gr</DocumentationURL>
        <Dependencies>manpages</Dependencies>
</package>
```

From the previous example, the virtual package depends on the physical one and one more package, that is part of the platform repository. Both are classified under the Computing capability, that although is not strictly correct (their function strictly speaking is to provide documentation for the computing capability provided by glite-WN), it makes sense as this is the capability they must support. Most of the remaining metadata are self-explanatory.

As we can see, there is no mention of the UMD Release that these packages are part of. This is intentional, as this decision will be made at the EGI Software Repository level and not by the software provider, who is also the metadata provider through the RT.

2.3.2 Option 2: Structured XML file

This provides the software provider with a way to define the structure of a release, along with the release and package metadata.

```
<release Name="UMD">
<Contact>eangelou@cslab.ntua.gr</Contact>
<Description>Yet another umd release</Description>
<DocumentationLinks>http://www.cslab.ece.ntua.gr http://www.egi.eu<DocumentationLinks>
<Version>1.0</Version>

<Capability Name="Computing">

<SoftwareComponent Name="glite3.2-WN">
<SoftwareProvider>EMI</SoftwareProvider>
```

<Contact>eangelou@cslab.ntua.gr</Contact>
<Description>Yet another glite WN release</Description>
<Keywords>gLite WN</Kewords>
<Date>21/6/2010</Date>
<DocumentationLinks>http://www.cslab.ece.ntua.gr http://www.egi.eu<DocumentationLinks>
<Version>3.2</Version>

<**package** Name="glite-WN-manpages" Type="Virtual" SLDType="community">
<Creator>eangelou</Creator>
<Contact>eangelou@cslab.ntua.gr</Contact>
<Description>Provides glite-WN with manpages for the appropriate tools</Description>
<License>GPL v.3</License>
<Keywords>gLite gLite-WN man manpages</Keywords>
<SupportedPlatforms>SL5</SupportedPlatforms>
<Architecture>any</Architecture>
<DocumentationLinks>http://www.cslab.ece.ntua.gr http://www.egi.eu<DocumentationLinks>
<Date>21/6/2010</Date>
<Version>3.2.1</Version>
<ReleaseNotesURL>http://www.cslab.ece.ntua.gr</ReleaseNotesURL>
<ChangeLogURL>http://www.cslab.ece.ntua.gr</ChangeLogURL>
<RepositoryURL>http://repository.egi.eu/</RepositoryURL>
<DocumentationURL>http://www.cslab.ece.ntua.gr</DocumentationURL>
<Dependencies>glite-WN-manpages-3.2.15 manpages</Dependencies>
</**package**>
<**package** Name="glite-WN-manpages-3.2.15" Type="Physical" SLDType="community" Method="apt">
<Creator>eangelou</Creator>
<Contact>eangelou@cslab.ntua.gr</Contact>
<Description>Provides glite-WN with manpages for the appropriate tools</Description>
<License>GPL v.3</License>
<Keywords>gLite gLite-WN man manpages</Keywords>
<SupportedPlatforms>SL5</SupportedPlatforms>
<Architecture>any</Architecture>
<DocumentationLinks>http://www.cslab.ece.ntua.gr http://www.egi.eu<DocumentationLinks>
<Date>21/6/2010</Date>
<Version>3.2.15</Version>
<ReleaseNotesURL>http://www.cslab.ece.ntua.gr</ReleaseNotesURL>
<ChangeLogURL>http://www.cslab.ece.ntua.gr</ChangeLogURL>
<RsyncURL>ftp://www.cslab.ece.ntua.gr/</RsyncURL>
<DocumentationURL>http://www.cslab.ece.ntua.gr</DocumentationURL>
<Dependencies>manpages</Dependencies>
</**package**>
</**SoftwareComponent**>
</**Capability**>
</**release**>

It is evident that this format provides more functionality to the software providers, as they can define the structure instead of just the package metadata in the release. This approach however introduces two further problems, namely, that the software provider or the RT knows in advance the structure of the UMD Release and that it can be altered irrespective of the EGI Repository team.

The previously outlined options are currently considered and since there are no incompatibilities between them they can both be supported simultaneously.

2.4 Functional requirements – workflow

The EGI Software Repository will have to support the workflow of the EGI Software release cycle that is described briefly bellow. A more detailed description of the workflow is given in [R 3].

Each time a new version of a component is released from the Software Providers its downloaded onto

the EGI Software Repository labelled “Upload”. After an automatic checksum verification the new released is transferred to the “Unverified” repository in order to be tested against the EGI Quality control criteria. If the tests are successful the new release is moved to the “Stage-Rollout” repository to be tested in real production environment by the early adopters. If the experimental deployment is satisfactory the new software is made available to the general public from the production repository. If any of the aforementioned steps fails the release is rejected and the whole process starts again with a new release.

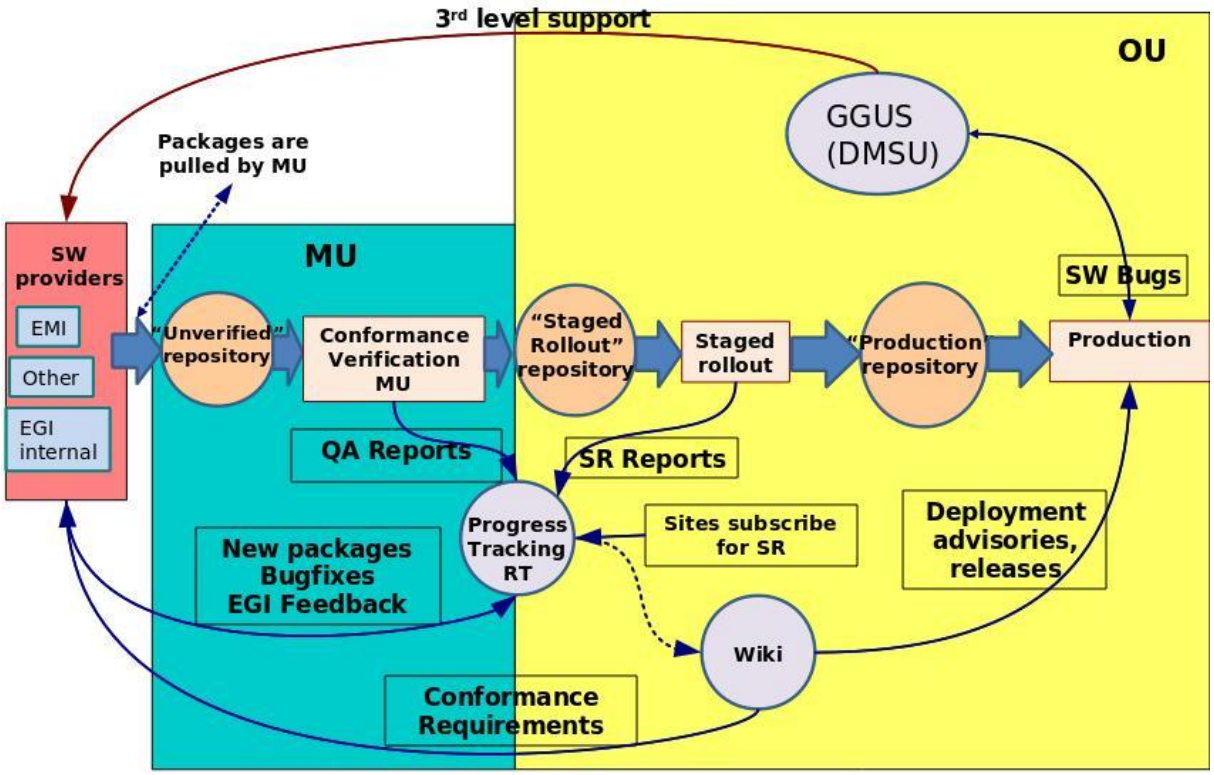

Figure 2: The EGI software Release cycle and actors.

2.5 Non-functional requirements

The infrastructure supporting the EGI Software Repository service needs to meet several standards in order to be reliable, robust and secure. Depending on the traffic and load requirements more than one server will be needed. These servers will be supplied with failover and/or load balancing procedures in order to achieve high-performance and high-availability. Also, backup and restore procedures should be defined and implemented to avoid data loss and to ensure fast recovery time. These procedures should be applied to metadata and depending to the size of the actual data could include some or all of them. A systematic system maintenance will keep the services secured from outside attacks. Finally, a monitoring system is needed to notify the maintainers for failures. Also, monitoring can measure and evaluate the capacity of the infrastructure.

3 ARCHITECTURE

3.1 Introduction

Due to the complex nature of the EGI software repository, consisting of many discrete components across various servers, a distributed architecture is needed; such a demand may be met by implementing a multi-tier client-server architecture, which will principally allow the separation of domain logic from data storage. Multiple modules in a presentation tier could also be provisioned for, in order to implement end-user interaction for the various user groups that may make use of the EGI Software Repository. (see **Error! Reference source not found.**). In such a case, we may simply speak of a three-tier architecture.

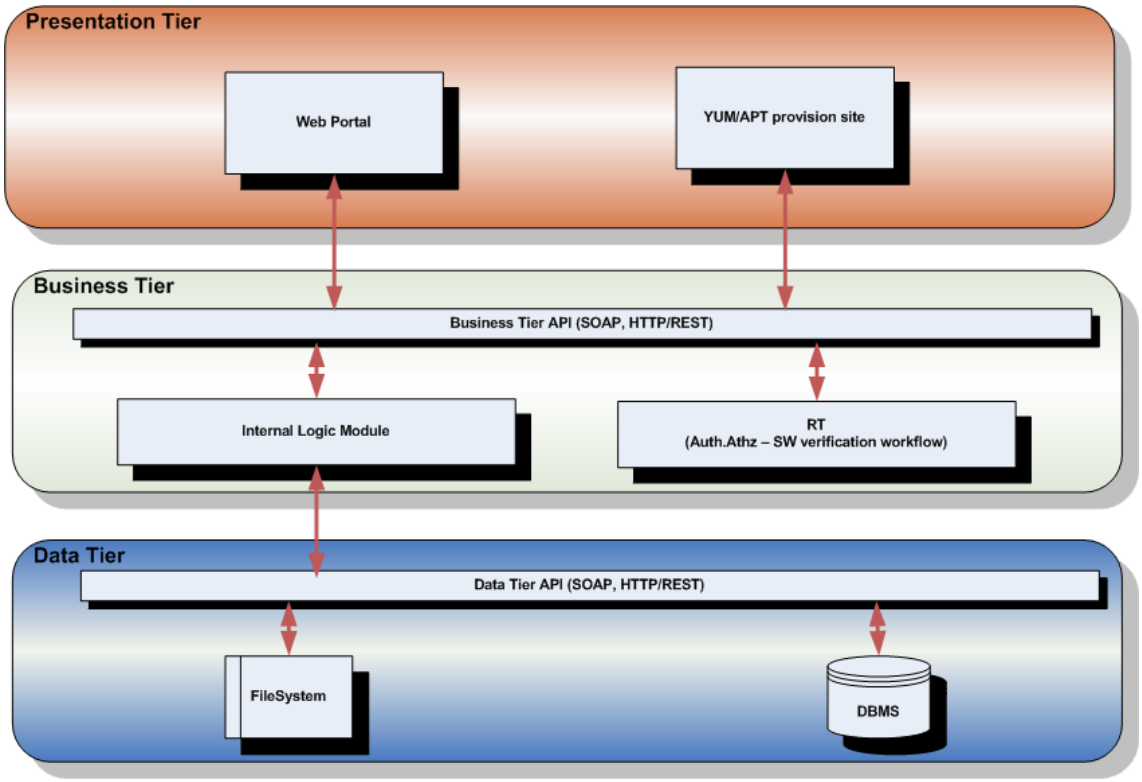

Figure 3: Three-tier system architecture diagram

Three-tier architecture is the most widely-spread case of multi-tier architecture, where the user interface, domain logic, and data storage/access are developed as separate supermodules, which may well reside on discrete physical systems and platforms, across a network. By thus segmenting development, one has the advantage of being able to replace or to upgrade any one of the tiers independently, as long as the programmatic interfaces are honoured. Moreover, one may deploy multiple tiers of the same identity in order to provide support for alternative scenarios, or even different systems, with rather minimal impact on development complexity.

3.2 Data Tier

3.2.1 Storage Backend

As is always the case, the choice of data storage backend is imposed on by the nature of the data to be stored. In the case of the EGI Software Repository, the main persistent data artefacts are binary files (packages). Storing that sort of data in a traditional DBMS – or any other usual kind of data repository – is most certainly a cumbersome task, which does not pay off well in terms of space or time complexity. The most natural choice in such cases is a filesystem, with an appropriate directory structure.

On the other hand, additional persistent data to be stored is metadata about the said physical artefacts, as well as for groups of those (virtual artefacts, e.g. a release or a metapackage), and so on. Quasi-transient metadata of a similar nature must also be kept in order to account for the state of subsets of the repository, during the course of such processes as the verification and evaluation of a new release. This kind of data is ideally stored in a DBMS.

Taking under consideration the following two observations, one may conclude that the best choice for a data storage backend would be a hybrid solution of a hierarchical filesystem scheme containing the primary artefacts, complemented by a DBMS, which will hold relevant metadata while pointing to physical location for each artefact, as well as metadata about virtual artefacts and repository states.

3.2.2 Data/metadata content

As mentioned in the previous paragraph, primary physical data artefacts are binary files in the form of packages. This includes, but is not limited to, RPM, DEB, and TAR files, the former intended for compiled code, while the latter mainly for source code packaging. Each of the above packages is described by a set of metadata attributes, conforming to the DTD implied by the XML example given in §**Error! Reference source not found.**.

As for virtual artefacts, metapackages are also described by the above set of attributes, plus extra attributes defining their members. Releases need a set of metadata conforming to the appropriate subset of the DTD implied by the relevant element of the XML example given in §**Error! Reference source not found.**. An overall example DTD for persistently stored metadata is provided in table 1.

```
<!ELEMENT release (Name, SoftwareProvider, SoftwareComponent, Contact, Description, Keywords, DocumentationLinks,
Date, Version, package+)>
<!ELEMENT Capability SoftwareComponent*>
<!ELEMENT SoftwareComponent SoftwareProvider, Contact, Description, Keywords, Date, Documentation, Version>
<!ELEMENT package (Name, SoftwareProvider, SoftwareComponent, Creator, Contact, Description, License, Keywords,
SupportedPlatforms, Architecture, DocumentationLinks, Date, Version, ReleaseNotesURL, ChangeLogURL,
DocumentationURL, RepositoryURL?, RsyncURL?, Category?, ServicesAffected?, Dependencies?)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT SoftwareProvider (#PCDATA)>
<!ELEMENT SoftwareComponent (#PCDATA)>
<!ELEMENT Creator (#PCDATA)>
<!ELEMENT Contact (#PCDATA)>
<!ELEMENT Description (#PCDATA)>
<!ELEMENT License (#PCDATA)>
```

```
<!ELEMENT Keywords (#PCDATA)>
<!ELEMENT SupportedPlatforms (#PCDATA)>
<!ELEMENT DocumentationLinks (#PCDATA)>
<!ELEMENT Date (#PCDATA)>
<!ELEMENT Version (#PCDATA)>
<!ELEMENT ReleaseNotesURL (#PCDATA)>
<!ELEMENT ChangeLogURL (#PCDATA)>
<!ELEMENT DocumentationURL (#PCDATA)>
<!ELEMENT RepositoryURL (#PCDATA)>
<!ELEMENT RsyncURL (#PCDATA)>
<!ELEMENT Category (#PCDATA)>
<!ELEMENT Architecture (#PCDATA)>
<!ELEMENT ServicesAffected (#PCDATA)>
<!ELEMENT Dependencies (#PCDATA)>
<!ATTLIST release
        Name CDATA #REQUIRED>
<!ATTLIST Capability
        Name CDATA #REQUIRED>
<!ATTLIST SoftwareComponent
        Name CDATA #REQUIRED>
<!ATTLIST package
        Name CDATA #REQUIRED
        Type (Virtual|Physical) #REQUIRED
        Method (yum|apt|tar)
        SLDType (community|full|associated) "community">
```

Table 1: Example DTD for persistent metadata

The process of deploying software into the EGI production infrastructure is described in detail in reference [R3], whence the set of quasi-transient metadata can be inferred. This includes verification status for individual certified packages (and/or metapackages), timestamps of staged-rollouts, early adopters identities, etc.

3.2.3 Data Tier interface

Access to data and metadata through a unified interface would certainly be a design asset, allowing simplification of the business tier. Hence, the need of an infrastructure, wrapping remote calls to DBMS' and filesystems accordingly, becomes apparent. Exposing a dual REST/SOAP interface seems to be a good choice for such an infrastructure, since REST copes well with handling collections of objects and SOAP works nicely with XML data. Furthermore, a REST/SOAP interface has the advantage of being accessible in a variety of ways, thus little limiting the choice of further technical development options, such as programming language compatibility, and remote communication protocols.

3.3 Business Tier

3.3.1 Web accessible API

The purpose of the business tier API is to provide simplified and filtered access to stored data, as well as to allow authorized users to add new and modify existing data. Since there might be multiple modules in the presentation tier, distributed over the internet, such an API must be web accessible, and therefore a dual REST/SOAP web API can, again, be used, thus keeping a certain degree of homogeneity across tiers.

The business tier API will also be accessed by the RT module belonging to the business tier, thus keeping a single line of data flow towards the data tier.

There is also some consideration about polling the RT system as an alternate means of communication. This option will be evaluated and in case it is deemed needed, in order to avoid technical communication difficulties with the RT module, it will be implemented at a later stage.

3.3.2 RT module

The RT module, will be responsible for orchestrating the SW verification process, by implementing the workflow described in [R3]. The workflow described in [R3] will be implemented over a dedicated staged-rollout queue in RT using custom fields within tickets to store relevant portions of the SW metadata. The metadata held for each SW package include:

- Certification status,
- Repository URL,
- Release notes,
- Changelog,
- External links.

RT scripts mechanism will be used to implement individual actions of the SW verification process including actions on the SW repositories. Examples of such actions would be:

- automated SW package handling in the staged-rollout repository once the SW metadata is verified in the RT,
- automated transfer of the SW package from the staged-rollout repository to production one once the SW verification process is closed and the respective RT ticket is successfully closed.

Requests for action on the repository will be processed through the business tier API, which will in turn trust the RT module, as the later will also be responsible for implementing any required user authentication and authorization. Nevertheless, user credentials will be sent to the data tier for reference storage.

3.3.3 Internal logic module

The internal logic module is the heart of the business tier. It is responsible for sanitizing requests to the data tier and retaining the data's logical integrity throughout operations. It interfaces directly to the data tier API and transforms data requested from (sent to) the business tier API into the form appropriate for presentation (storage) accordingly.

Examples of internal actions that must be carried out by the module are

- the creation of a new release under the staged-rollout area upon submission of a new release for verification,
- moving a release from the stage-rollout area to the production area upon receiving notification of a new release's verification process completion,

- requesting the data tier to log user credential information upon data modification requests,
- rejecting data modification requests that do not comply with the process described in R3, etc..

3.4 Presentation TIER

3.4.1 WebPortal

The EGI repository portal provides a centralized point to search for software developed for the EGI project. It will support end-users who want to search for new software releases or individual packages. Currently, this goal is met manually from the UMD Repo Team. A preliminary web portal has been setup, that contains the current production releases of the most used grid middleware software within the EGI project, namely gLite versions 3.2 and 3.1, ARC and UNICORE. The site provides an easy way to search for specific packages of the supported releases, preliminary documentation for installation and contact points or links to further documentation. Users have at their disposal most of the documentation available from a single, consistent interface.

Finally, the goal is to integrate the portal with the software reviewing process, thus providing more features to both technology providers and reviewers. For more information, please refer to [R1].

3.4.2 Repository provision site

The repository provision site provides access to UMD's YUM/APT package repositories, by means of web browsing for end-users, and by acting as an access point for the automated package installation tools of the supported OS distributions. A preliminary version can be accessed at the location specified by [R4], where packages are classified according to status (e.g. production, stage-rollout), then by provider (e.g. EMI), and then by consortium (e.g. gLite).

4 NEXT STEPS – PLANS

In order to best route development effort, the use of some third-party open source solutions as part of the implementation will be considered. At the business tier, the use of a workflow system such as Taverna [R 5] or Knime [R 6] may be used in the Internal Logic module, both of which are capable of executing complex workflows and making REST and SOAP calls. The use of such a system has the advantage of visually depicting process and information flow, making it more intuitive, and thus speeding up development and making debugging easier.

At the data tier, using a repository solution such as Fedora Commons [R 11] to store metadata, instead of a naked DBMS, may be a valuable asset. More specifically, the eSciDoc [R 7] core services infrastructure will be deployed and evaluated. It encapsulates a Fedora Commons repository and implements a broad range of commonly used functionalities on top of it. Its service-oriented architecture fosters the creation of autonomous services, which can be re-used independently from the rest of the infrastructure. Some of the key features of the eSciDoc core services are:

- Flexible content models
- Arbitrary metadata profiles
- Application-independent design
- Hierarchies and networks of Organizational Units
- Support for object relations and multiple ontologies
- Statistics
- Dual SOAP/REST API

The EGI Software Repository development team will be in close contact with the EGI shareholders during these early stages of development, providing feedback on the evaluation procedure of said architecture and technologies, while listening in for changes and new demands in needs and requirements of the project.

Table 2: Next Steps.

Step	Project Month	Description	Status
1	**M1**	Web Interface + Mirrored APT/YUM Repositories	Complete
2	**M6**	Implementation of the Software Release Workflow and Integration with RT	In Progress
3	**M7**	Validation of the workflow in collaboration with JRA1	In Progress
4	**M7-M12**	Continuous and iterative improvement of the SW Release Workflow towards handling an integrated EMI release.	Scheduled.

5 CONCLUSIONS

This document describes the current requirements and the system architecture defining the EGI Software Repository and associated support tools. Chapter 3 analyses the requirements gathered for a reliable and functional EGI Software Repository. Within the same section a common vocabulary and schema to be used for the interaction with the software providers, it is established.

Due to the complex nature of the EGI software repository, consisting of many discrete components across various servers, a distributed architecture is needed; such a demand may be met by implementing a multi-tier client-server architecture, which will allow the separation of domain logic from data storage. The presentation-tier comprises of a Web-Portal that serves as a centralized point for the users to search for software developed for the EGI project and by the repository provision portal which aims to offer a mechanism automated package installation (APT/YUM). The domain logic (business tier) includes, an internal logic module, responsible for sanitizing requests to the data tier and an RT module, which is utilised to orchestrate the software release cycle. The data storage/access (data tier) which is a hybrid solution of a hierarchical file system scheme containing the primary artefacts, complemented by a DBMS, which will hold relevant metadata while pointing to physical location for each artefact, as well as metadata about virtual artefacts and repository states. TSA 2.4 is at the time of writing in negotiation with the Technology providers in order to define the last minor technical details needed to start the implementation phase for the next release of the EGI Software Repository and Associated Support Tools. As its usual the case during the installation phase both the requirements and System Architecture of the EGI software repository will have to be adapted to both the needs of the user community it is targeted to serve and the outcome of the negotiations with the technology providers on how to receive new software releases from them. To cater for these, the plan for the next year we have scheduled a number of iterative releases in M7 to M12 that will address any bugs or limitation of the system found during the implementation and testing phase. These foreseen will be documented with yearly updates of this milestone that are scheduled for the duration of the EGI-InSPIRE project.

.

6 REFERENCES

R 1	MS501 - Establishment of the EGI Software Repository and associated support tools https://documents.egi.eu/document/46
R 2	The Portal of the Repository: http://repository.egi.eu/
R 3	MS402 - Deploying software into the EGI production infrastructure https://documents.egi.eu/document/53
R 4	The Repo view: http://repository.egi.eu/sw
R 5	Taverna Workflow Management System: http://www.taverna.org.uk/
R 6	Knime Workflow Management System: http://www.knime.org/
R 7	eSciDoc - The Open Source e-Research Environment: https://www.escidoc.org/
R 8	MS402: Deploying Software into the EGI Production Infrastructure
R 9	MS503: Software Provisioning Process
R10	D5.1 UMD Roadmap https://documents.egi.eu/public/ShowDocument?docid=100
R 11	Fedora Commons : http://fedora-commons.org/