



# EGI-InSPIRE

## EGI SOFTWARE REPOSITORY ARCHITECTURE AND PLANS

### EU MILESTONE: MS506

---

Document identifier:	EGI-InSPIRE-MS506-v1-reviewed.docx
Date:	<b>14/06/2011</b>
Activity:	<b>SA2</b>
Lead Partner:	<b>GRNET</b>
Document Status:	<b>FINAL</b>
Dissemination Level:	<b>PUBLIC</b>
Document Link:	<a href="http://documents.egi.eu/document/503">http://documents.egi.eu/document/503</a>

---



### 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 2 of the EGI-InSPIRE Project.

## I. COPYRIGHT NOTICE

Copyright © Members of the EGI-InSPIRE Collaboration, 2010. See [www.egi.eu](http://www.egi.eu) for details of the EGI-InSPIRE project and the collaboration. EGI-InSPIRE (“European Grid Initiative: Integrated Sustainable Pan-European Infrastructure for Researchers in Europe”) is a project co-funded by the European Commission as an Integrated Infrastructure Initiative within the 7th Framework Programme. EGI-InSPIRE began in May 2010 and will run for 4 years. This work is licensed under the Creative Commons Attribution-Noncommercial 3.0 License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc/3.0/> or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, and USA. The work must be attributed by attaching the following reference to the copied elements: “Copyright © Members of the EGI-InSPIRE Collaboration, 2010. See [www.egi.eu](http://www.egi.eu) for details of the EGI-InSPIRE project and the collaboration”. Using this document in a way and/or for purposes not foreseen in the license requires the prior written permission of the copyright holders. The information contained in this document represents the views of the copyright holders as of the date such views are published.

## II. DELIVERY SLIP

	Name	Partner/Activity	Date
From	K. Koumantaros	GRNET/SA2	15/5/2011
Reviewed by	<b>Moderator:</b> Karolis Eigelis <b>Reviewers:</b> Ales Krenek	EGI.eu/NA3 CESNET	6/6/2011
Approved by	<b>AMB &amp; PMB</b>		14/6/2011

## III. DOCUMENT LOG

Issue	Date	Comment	Author/Partner
1	2/5/2011	First draft	K.Koumantaros/GRNET
2	4/5/2011	Adding contributions from M.Kuba, M. Liška and M. Chatziangelou	M.Kuba&M.Liška/CESNET M. Chatziangelou/GRNET
3	5/5/2011	2nd Revision.	K.KOUMANTAROS/GRNET
4	6/5/2011	Adding contributions in sections 2 & 3 overall revision.	M. Chatziangelou/GRNET C. Boumpouka/GRNET
5	7/5/2011	Adding contributions in sections 2 & 3	C. Theodosiou/GRNET
6	9/5/2011	addressing and comment from M. Drescher	K. Koumantaros et al/GRNET
7	16/6/2011	Version Ready for External Review	K. Koumantaros / GRNET

## IV. APPLICATION AREA

This document is a formal milestone 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/about/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 design and architecture of the EGI Software Repository (<http://repository.egi.eu>) and associated support tools. Section 2 is dedicated to presenting in detail the different components that are involved in the Unified Middleware Distribution (UMD) provisioning workflow. The first two subsections refer to the workflow of a new software release bundle into the EGI Software Repository and the procedures that internal and external Technology Providers have to follow in order to submit such a new release. The rest of the section deals with what a UMD release is, as well as describing the necessary metadata that should escort releases in order to be successfully submitted to the RT and therefore qualify for a UMD release. The metadata that are provided to the RT are stored to the EGI Repository and are used locally to provide meaningful semantics to the users. Section 3 describes the current implementation of the aforementioned processes using EGI's repository and associated support tools. Finally section 4 discusses future plans for the 2nd year of the EGI-InSPIRE project. This document is a formal milestone 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. It is of most interest for the Technology providers, and EGI WP4 (SA1) Release management.

## TABLE OF CONTENTS

<b>1</b>	<b>INTRODUCTION .....</b>	<b>8</b>
<b>2</b>	<b>UMD PROVISIONING WORKFLOW .....</b>	<b>9</b>
2.1	Introduction .....	9
2.2	Internal technology providers Software Provisioning Process .....	9
2.3	External technology providers Software Provisioning Process .....	10
2.4	UMD Release building .....	11
2.5	Internal Technology Provider metadata .....	12
2.6	External Technology Provider metadata .....	13
<b>3</b>	<b>ARCHITECTURE .....</b>	<b>15</b>
3.1	Introduction .....	15
3.2	Data Layer .....	15
3.2.1	Storage Backend .....	15
3.2.2	Metadata backend & content .....	18
3.3	Business Layer .....	18
3.3.1	Bouncer Component .....	18
3.3.2	Request Tracker (RT) Component .....	18
3.3.3	Repo-backend Component .....	21
3.3.4	UMD Composer .....	25
3.4	External interaction layer .....	28
3.4.1	Web Portal .....	28
3.4.2	Repository Provision Site .....	29
3.4.3	GGUS Component and Dashboard .....	30
3.5	Auxiliary Components .....	32
3.5.1	Replication Mechanism .....	32
3.5.2	Metrics Collection .....	33
<b>4</b>	<b>NEXT STEPS – PLANS .....</b>	<b>34</b>
<b>5</b>	<b>REFERENCES .....</b>	<b>35</b>
5.1	Table of Figures .....	36
<b>6</b>	<b>APPENDIX .....</b>	<b>37</b>



## 1 INTRODUCTION

This document thoroughly presents the architecture and functionality of the EGI Software Repository (<http://repository.egi.eu>), along with the tools developed around it. Important aspects of the Repository are argued, such as the definition of a *Unified Middleware Distribution*, also known as a *UMD release*, what kind of software bundles qualify for a *UMD release*, the submission procedure for Technology Providers, as well as the workflow that submitted packages will have to pass through before ending up in an official *UMD release*. The Repository's architectural design is explained and the behaviour of each component participating in the submission process is analysed. The remainder of the milestone is organized as follows: Section 2 describes EGI's Software Provisioning process for both internal and external Technology Providers. Section 3 describes the current implementation of the aforementioned processes using EGI's Repository and associated support tools. Finally, section 4 discusses future plans for the 2nd year of the EGI-InSPIRE project. This document is a formal milestone 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. It is of most interest for the Technology providers, and EGI WP4 (SA1) Release management.



## 2 UMD PROVISIONING WORKFLOW

### 2.1 Introduction

This section is dedicated to presenting in detail the different components that are involved in the Unified Middleware Distribution provisioning workflow. The first two subsections refer to the workflow of a new software release bundle into the EGI Software Repository and the procedures that internal and external Technology Providers have to follow in order to submit such a new release. Software releases are submitted via the Request Tracker (RT) ticketing system, introduced in Section 3.3.2. The rest of this section deals with what a *UMD release* is, as well as describing the necessary metadata that should escort releases in order to be successfully submitted to the Request Tracker (RT) and therefore qualify for a *UMD release*. The metadata that are provided to the RT are stored to the EGI Repository and are used locally to provide meaningful semantics to the users. The proposed metadata schema, outlined in Sections 2.5 and 2.6 for the Internal and External Technology Providers respectively, is rich enough to allow all parts of the repository infrastructure (YUM/APT repositories and the web portal) to extract the necessary information about the different software releases.

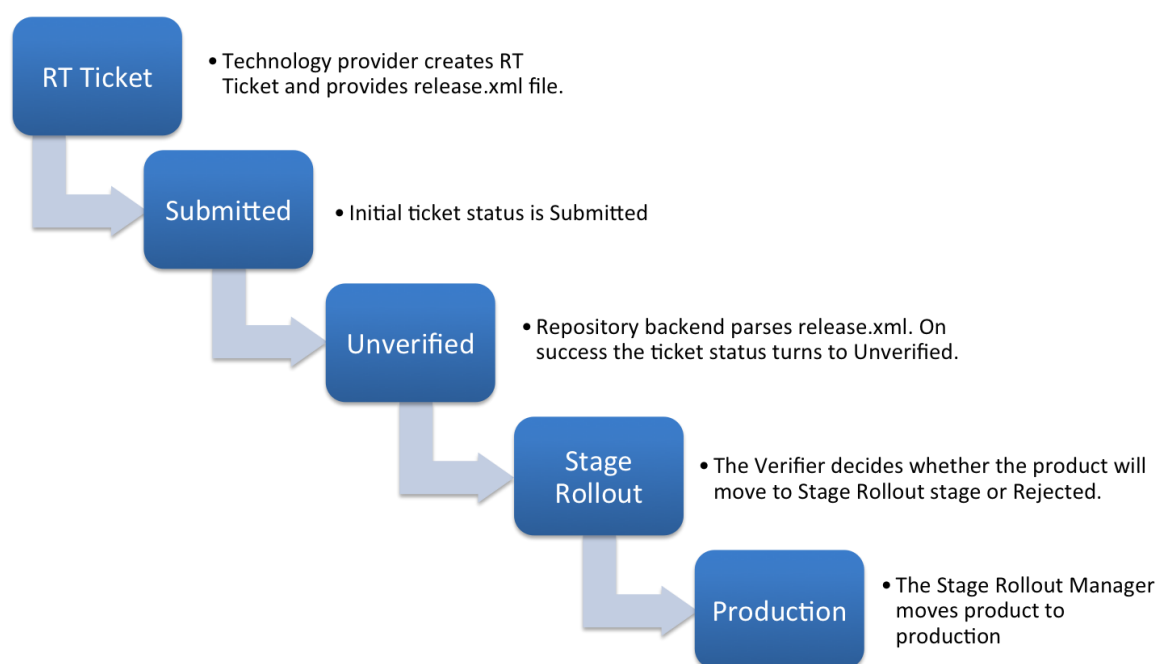
### 2.2 Internal technology providers Software Provisioning Process

The release workflow of new software, depicted in Figure 1 for internal Technology Providers, can be viewed and managed by the Request Tracker (RT) component of EGI (<http://rt.egi.eu>). Additionally, automation procedures are handled by a backend component of the repository (<https://admin-repo.egi.eu>), referred to as *Repo* in the remainder of this document. These procedures are responsible for the creation and management of the software. However, the triggering and execution of these procedures are entirely managed through RT.

The process starts upon technology provider upload of a new *release.xml* file to RT. This file contains all the information needed. A detailed description of the structure of *release.xml* file can be found in section 2.5. A new ticket is created in RT with a custom field named **RolloutProgress**. The value of the field is **Submitted**. The backend starts with the parsing of the *release.xml*, it then downloads the software bundle and puts it on the “Unverified” Repository. If this procedure encounters an error, the **RolloutProgress** will remain to “Submitted” and the RT-Repo status will be set to **Failed**. Additionally, it will update the ticket in order to notify the queue watchers. If the parsing procedure is successful then **RolloutProgress** is set to **Unverified**.

Afterwards, the ticket is assigned to the Software Verifier. If any issues arise during verification, the Technology Provider is notified. The issues should be either resolved or clarified within a specified time period. Otherwise, the ticket is set to **Rejected** and the *Repo* backend moves the software to the “Rejected” repository. When the verification process is completed the Software Verifier sets the **RolloutProgress** field to **StageRollout**. This action triggers a new procedure in the backend, which moves the software to the “StageRollout” repository.

Moreover, RT notifies the Early-Adopters, that a new software product has been moved into the “StageRollout” area of the repository. At this stage, the software should follow the “StageRollout” procedure and upon completion, regardless of the outcome, the Stage-Rollout Manager<sup>1</sup> is responsible of changing the *RolloutProgress* to *Production*, if the procedure was successful, or to *Rejected* state otherwise. This action will trigger the backend’s procedure that moves the software to the appropriate area of the repository.



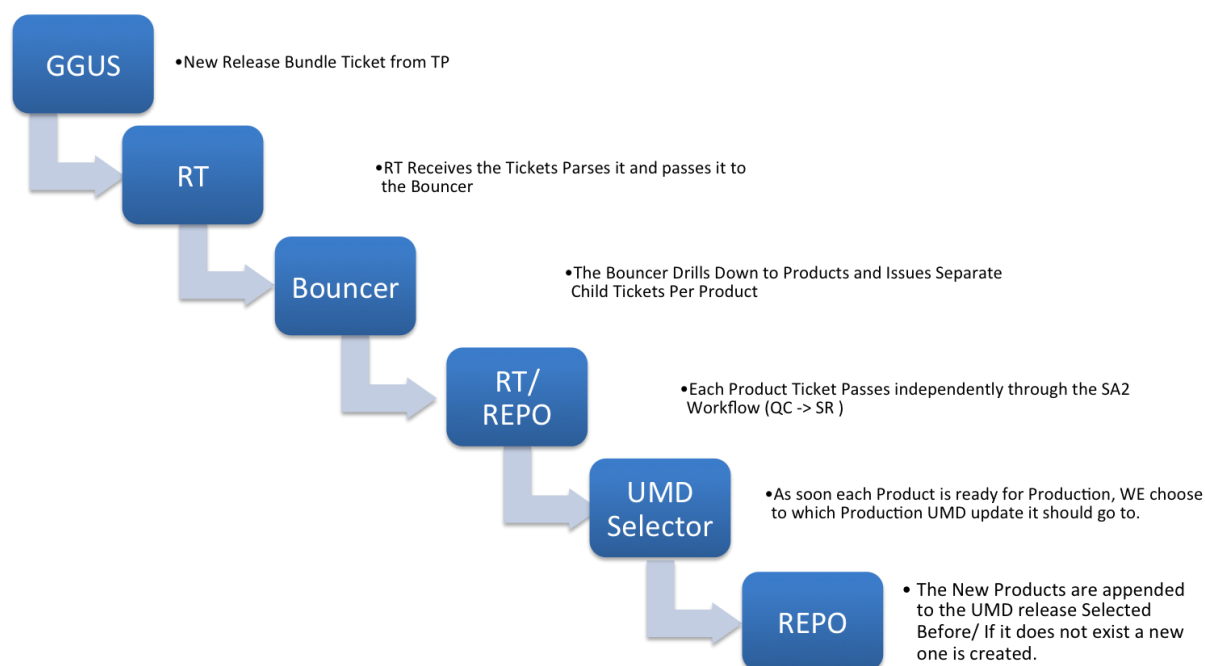
**Figure 1 Internal Technology Providers Provisioning Process**

## 2.3 External technology providers Software Provisioning Process

The release workflow of an external Technology Provider, depicted in Figure 2, is similar to the one described before, incorporating some additional steps. First of all, external Technology Providers use the GGUS ticketing system for submitting new releases. Since the software bundles of external Technology Providers may not conform to the structure required by the EGI Repository, a slightly different process is applied when processing releases from external Providers. The purpose of the extra steps in the process is to transform external software bundles submitted to GGUS into appropriate RT tickets that can be process by the *Repo*. The release procedure begins with a GGUS ticket, which includes a *release.xml* file describing the software bundle under release. In order to

<sup>1</sup> The person responsible to oversee EGI InSPIRE TSA1.3 task Service Deployment and Validation

describe the entire bundle, this *release.xml* file has an extended structure. The structure of the file is described in detail in section 2.6. GGUS triggers an RT ticket similar to EGI's RT tickets. Afterwards; RT passes the ticket to the *Bouncer* component, presented in Section 3.3.1. The bouncer is responsible for dividing the release into several Software Products<sup>2</sup>. Each product is assigned a separate RT ticket. From this point on, the release process follows the release process of internal software products. Each product is treated as if it was an internal Technology Provider's product. The only difference is that the accepted products are not moved directly into the 'Production' area. They are first moved to an intermediate area, called "UMDStorerepository", where the *UMD Composer* component, described in Section 3.3.4, takes over. By using the *UMD Composer* component, the UMD Release Manager is authorized to 'compose' a *UMD release* with a given set of products and finally to deploy the release into the 'Production' repository.



**Figure 2 External Technology Providers Software Provisioning Process**

## 2.4 UMD Release building

<sup>2</sup> A Product is a solution delivered by Technology Providers to EGI and provides the functionality for one, sometimes more Capabilities as one single, undividable unit.

EGI follows its own release schedule, publishing fixed UMD updates (monthly/quarterly). Accepted PPAs (Product per Platform and Architecture) versions are stored into a staging area of the EGI Repository as delivered by the Technology Providers. When a new UMD release is to be created, the UMD Release building process collects the qualifying and compatible with each other Products from the EGI Repository and bundles them into UMD releases. That is, a UMD release is irrespective of the Technology Providers releases.

There are two types of UMD Releases:

- Base, a new major release, that includes backwards incompatible updates.
- Update, a subsequent minor or revision release

## **2.5 Internal Technology Provider metadata**

In order for the repository to process a new software release bundle, an xml metadata document (i.e. *release.xml*) providing all necessary information is required. The accompanying file contains necessary information about the software, structured in six major logical sections:

- The general release section
- The UMDmeta section (ONLY for products that are candidates for a UMD release, that is products that have successfully passed through the UMD workflow)
- The YUM Repositories section
- The APT Repositories section
- The YUM repofiles section and
- The APT repofiles (i.e. “.list” files) section.

Each of these sections is defined by an xml element with the appropriate fields. A high level view of such an XML file is shown in Figure 3. A detailed description of each section’s fields can be found in [R 20].

```
- <release:release xsi:noNamespaceSchemaLocation="release.xsd">
  <release:productName>EGI Core Trust Anchor Distribution</release:productName>
  <release:productShortName>cas</release:productShortName>
  <release:technologyProvider>European Grid Initiative</release:technologyProvider>
  <release:technologyProviderShortName>EGI</release:technologyProviderShortName>
  <release:contact>egi-igtf-liaison@nikhef.nl</release:contact>
  <release:technicalContact>egi-igtf-liaison@nikhef.nl</release:technicalContact>
  <release:description>Trust anchors endores by the EGI.eu foundation</release:description>
  <release:documentationLink>https://wiki.egi.eu/wiki/EGI_IGTF_Release</release:documentationLink>
  <release:documentationLink>https://wiki.egi.eu/wiki/EGI_IGTF_Release_Process</release:documentationLink>
  <release:keyword>ca-policy-egi-core</release:keyword>
  <release:keyword>trust</release:keyword>
  <release:keyword>CA</release:keyword>
  <release:keyword>IGTF</release:keyword>
  <release:ISODate>20110207</release:ISODate>
  <release:incremental>false</release:incremental>
  <release:emergency>false</release:emergency>
  <release:majorVersion>1</release:majorVersion>
  <release:minorVersion>38</release:minorVersion>
  <release:revisionVersion>1</release:revisionVersion>
+ <release:releaseNotes></release:releaseNotes>
- <release:changeLog>
  Update of the EGI Trust Anchors to IGTF release 1.38-1
  </release:changeLog>
  <release:synchProtocol>rsync</release:synchProtocol>
  <release:synchURL>rsync://egi-igtf.ndpf.info/egi-igtf/egi/</release:synchURL>
  <repository:yumRepository id="repo-ca" external="false" path="current"/>
+ <repofile:yumRepofile filename="egi-trustanchors.repo" relativePath="repofiles"/></repofile:yumRepofile>
</release:release>
```

Figure 3: Internal Technology Provider Release example.

## 2.6 External Technology Provider metadata

External Technology Providers need to produce a slightly different xml metadata document (aka *release.xml*) to escort their software distribution into the EGI Repository. A meaningful *release.xml* file consists of two major logical sections:

- The general release section and
- The delivered product(s) specific section

Each of these sections is defined by an xml element with the appropriate fields. A high level view of such an XML file is shown in Figure 4 and an extensive presentation of each section's fields can be

```

- <release:release xsi:noNamespaceSchemaLocation="release-ext-tp.xsd">
  <release:technologyProvider>European Middleware Initiative</release:technologyProvider>
  <release:technologyProviderShortName>EMI</release:technologyProviderShortName>
  <release:contact>youremail@here</release:contact>
  <release:technicalContact>youremail@here</release:technicalContact>
  <release:ISODate>20110210</release:ISODate>
  <release:emergency>false</release:emergency>
  <release:majorVersion>1</release:majorVersion>
  <release:minorVersion>2</release:minorVersion>
  <release:revisionVersion>3</release:revisionVersion>
- <product:product>
  <product:productName>Computing Resource Execution And Management</product:productName>
  <product:productShortName>cream-ce</product:productShortName>
  <product:description>Product description here ... </product:description>
  <product:documentationLink>http://cream.eu-emi.eu/userguide</product:documentationLink>
  <product:documentationLink>http://cream.eu-emi.eu/adminguide</product:documentationLink>
  <product:majorVersion>10</product:majorVersion>
  <product:minorVersion>22</product:minorVersion>
  <product:revisionVersion>30</product:revisionVersion>
  <product:releaseNotes>Product release notes here ... (Multiline)</product:releaseNotes>
  <product:changeLog>Product change logs here ... (Multiline)</product:changeLog>
  <product:capability>Job Execution</product:capability>
  <product:capability>Messaging</product:capability>
- <product:target platform="sl5" arch="i386">
  <product:metaPackage>glite-CREAM</product:metaPackage>
  <product:updatedPackage>...</product:updatedPackage>
  <product:updatedPackage>...</product:updatedPackage>
  <product:updatedPackage>...</product:updatedPackage>
  <product:updatedPackage>...</product:updatedPackage>
  </product:target>
- <product:target platform="sl6" arch="x86_64">
  <product:metaPackage>glite-CREAM</product:metaPackage>
  <product:updatedPackage>...</product:updatedPackage>
  </product:target>
- <product:target platform="ubuntu-10.10" arch="amd64">
  <product:metaPackage>glite-CREAM</product:metaPackage>
  <product:updatedPackage>...</product:updatedPackage>
  </product:target>
</product:product>
</release:release>

```

found in [R 21].

**Figure 4: External Technology Provider Release example.**

## 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-layered architecture, which will principally allow the logical separation of the domain logic (Business Layer) from data storage (Data Layer). Multiple modules in an External Interaction Layer are also provisioned for, in order to implement end-user interaction for the various external user groups that may make use of the EGI Software Repository.

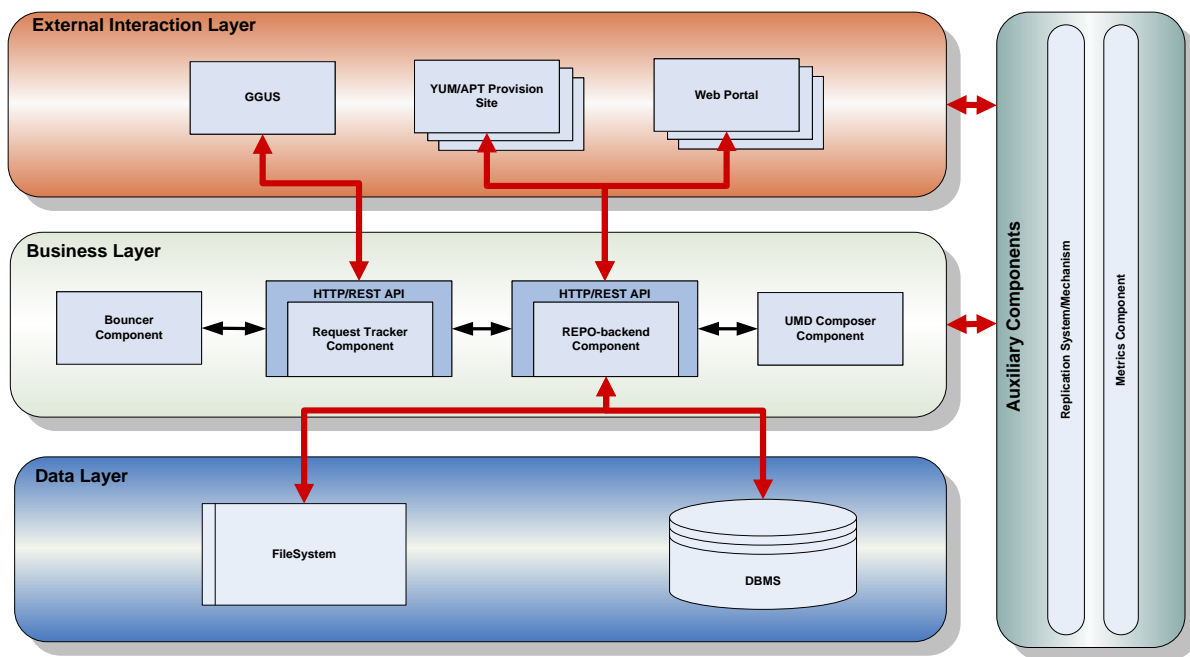


Figure 5: EGI Repository architecture diagram

### 3.2 Data Layer

#### 3.2.1 Storage Backend

The main persistent versioned data artefacts (also called product releases) that are stored in the EGI Software Repository are bundles of binary files (packages). Storing that sort of data in a traditional DBMS – or any other usual kind of data repository for that matter – is most certainly a cumbersome task, which does not pay off well in terms of space and time complexity. The most natural choice in



such cases is a file system, with an appropriate directory structure.

In addition, persistent data to be stored are metadata of the said physical artefacts, their groups/bundles, 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 processes such as the verification and evaluation of a new release. This kind of data is indeed ideally stored in a DBMS.

Taking into consideration the above-mentioned observations, we concluded that the best choice for a data storage backend 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 a physical location for each artefact or bundle, as well as metadata and repository states.

Based on this decision, six areas/directories were made available through the repository backend structure respectively to the repository states that are identified by the software provisioning workflow (see paragraph2 for more details):

**unverified:** Holds the bundle of binary artefacts that constitute the submitted product release which is under quality verification process. The file system structure of this area is following the pattern:

```
..sw/unverified/<product-short-name>/<Major-number>/<Minor-number>/<Revision-number>/os/arch
```

**stagerollout:** Holds the bundle of binary artefacts that constitutes the product release which is under stage-rollout procedure. The file system structure of this area is following the pattern below.

In **StageRollout** area, the Minor and Revision number are not used within the file system structure, this is because; (a) only one release per major of a given product could be available under the **StageRollout** and (b) having a file system structure that depends on the release minor and revision numbers, the Early-Adapter sites would be forced to adjust the YUM or APT related repofiles on every, frequently changing, minor or revision release of a product under the **StageRollout** phase, thus causing significant unnecessary delays to the whole workflow.

```
..sw/stagerollout/<product-short-name>/<Major-number>/os/arch
```

**deferred:** Holds the bundle of binary artefacts that constitutes the product release which is stated as **Deferred** (for more details on the **Deferred** state see the description of the Software Provisioning Workflow in section 3.2). The file system structure of this area is following the pattern:

```
..sw/deferred/<product-short-name>/<Major-number>/<Minor-number>/<Revision-num>/os/arch
```

**production:** Holds the final snapshot of a given product major release. This snapshot includes 'all' the releases (or the 'last', in case of a non-incremental release provisioning WF followed) of the



respective product that successfully passed the stage-rollout phase. The file system structure of this area is following the pattern below.

In **Production** area, the Minor and Revision numbers are not used within the file system structure. This is because the production area offers either the latest product release (in case of a non-incremental workflow followed) or the final snapshot of the product major release (in case of an incremental workflow followed). Additionally, having a file system structure that is depending on the release minor and revision numbers, the end-users/sites would be forced to adjust the YUM or APT related repofiles on every, frequently changing, minor or revision release of a product under the **StageRollout** phase, thus causing significant unnecessary delays to end-users.

```
..sw/production/<product-short-name>/<Major-number>/os/arch
```

**umdstore**: Holds the bundle of binary artefacts that constitute the submitted product release which is to be included in a given UMD release and has successfully passed the **StageRollout** phase. The file system structure of this area is following the pattern:

```
..sw/umdstore/<product-short-name>/<Major-number>/<Minor-number>/<Revision-num>/os/arch
```

**rejected**: Holds the bundle of binary artefacts that has been rejected during the Software Provisioning Workflow. The file system structure of this area is following the pattern:

In addition to the above, two complementary areas that do not participate directly in the software provisioning workflow are also available through the repository backend structure:

```
..sw/rejected/<product-short-name>/<Major-number>/<Minor-number>/<Revision-num>/os/arch
```

**mirrors**: Provides a collection of mirrors of *rsync* sites that the EGI and further on, the SA2 activity, has decided to make available through the EGI Repository. This area can be managed only by a group of authorized SA2 members using the Repository mirroring facility module (see section 3.3.3 for more details). This area is available at:

```
..mirrors/
```

**early-access**: Holds the bundle of the artefacts associated with the versioned product releases, which have been successfully passed into the Stage-Rollout phase; they are targeting a specific UMD-Major release, but they are not yet released into the production area.

The file system structure of this area is following the pattern:

```
..sw/early-access/UMD/<UMD-Major-number>/<Supported-OS(s)>/<Supported-ARCH(s)>/
```

### 3.2.2 Metadata backend & content

As mentioned in the previous paragraph, the versioned bundle of data artefacts (also known as product releases) are constituted by binary files in the form of packages. This includes, but is not limited to, RPM and DEB files. Each of the above bundles of packages is described by a set of metadata attributes, as those provided by the Technology Provider, conforming to the description provided in paragraph 3.5, as well as to the “Guide for internal technology provider” document [R19].

Moreover, additional metadata are needed to fully describe a given product release within the EGI Repository workflow (see section 3.2). These should provide details such as the latest state of the product release, URLs to the quality verification and stage-rollout reports, flags indicating whether a product release it is a block-listed one, insertion and last update timestamps, etc.

An autonomous database has been implemented, based on MySQL engine, in order to store and make the required metadata accessible for further use. Figure 13 in Appendix provides the Entity Relationship Diagram. A detailed description of the implemented database will be provided in a separate document (once the developments towards this direction are finalized), since such technical details are out of the scope of this document.

## 3.3 Business Layer

### 3.3.1 Bouncer Component

The bouncer is implemented as a Python script, processing two input parameters: a parent RT ticket number and an XML file with a release description from an external provider (the *release.xml*). The bouncer parses the release description, where a release may contain multiple products and each product may contain multiple target platforms. For each target platform it performs the following:

- Extracts all “metaPackages” and “updatedPackages” from the release description
- Using *yumdownloader* and *rpm* finds all dependencies in a remote repository
- Generates a new XML file containing information from the release description and a list of the found dependencies
- Creates a new RT ticket in the “sw-rel” queue as a child ticket of the specified parent ticket
- Attaches the generated XML as a “ReleaseMetadata” custom field of the RT ticket

### 3.3.2 Request Tracker (RT) Component

RequestTracker (RT) [R 13] is an enterprise-grade issue tracking system. It allows organizations to keep track of what needs to get done, who is working on which tasks, what's already been done, and

when tasks were (or weren't) completed. RT is used (but not limited to) for bug tracking, help desk ticketing, customer service, workflow processes or network operations. RT is provided as open source under the terms of version 2 of the GNU General Public Licence and allows for a customizability in a large extent. Namely, RT allows to categorize tickets and projects using the RT queues however makes sense for some particular purpose. All necessary workflow and business logic can be taught to RT through "scripts"<sup>3</sup>, lifecycles, custom fields, approvals, and extensions.

### 3.3.2.1 Design and Scripts

The software provisioning workflow is modelled using three RTR queues, a number of custom fields to store the status of the software through its provisioning process and RT scripts to provide the necessary automation to the software provisioning process. The software provisioning workflow is implemented through a set of mutually linked tickets in three different EGI RT queues breaking down the software provisioning process into logically decoupled sub processes.

If an external technology provider submits a software release, then a ticket is created in the *sa2-umd-rel* queue through the RT – GGUS interface described below. The release metadata is stored within a *ReleaseMetadata* custom field of upload one file type. Furthermore, the state of the ReleaseMetadata processing is held within the *SA2UMDRelStatus* custom field. The following states are allowed: *New*, *InBouncer*, *Pending*, *Finished*. Operations over the tickets in the *sa2-umd-rel* queue are mostly automated. The most important ones include checking and setting a default custom fields values on ticket creation, validation and processing of the release metadata provided by the external technology providers. The EGI Repository syntactically validates the release metadata using the XML validation API provided. If the validation is successful then the release metadata are passed to the Bouncer component of the workflow for further processing, otherwise an error is logged within the RT ticket.

The majority of the software provisioning workflow in the EGI RT is implemented within the *sw-rel* queue. The aim of this queue is to process particular product-platform-architecture (PPA) combinations of the software release. A new ticket is created for each PPA combination in the *sw-rel* queue. The ticket is created either automatically by the Bouncer (through the GGUS – *sa2-umd-rel* – Bouncer workflow) in case of a software provided by external technology providers or manually by internal technology providers. The release metadata for the PPA is again stored within the *ReleaseMetadata* custom field. The progress of the PPA processing is registered within the *RolloutProgress* custom field. Allowed progress states are: *Submitted*, *Unverified*, *InVerification*, *Waiting for response*, *Deferred*, *StageRollout*, *Production*, *UMDStore*, *Rejected* and *Ignored*. The following table depicts allowed state changes.

---

<sup>3</sup> Scripts: Scripts are the action driven scripts used to augment and customize the functionality within RT

Rollout Progress Allowed State Transitions								
RolloutProgress								
To state →	Unverified	In Verification	Waiting for response	StageRollout	UMDStore	Deferred	Production	Rejected
From state ↓								
Submitted	Trigger repo							
Unverified								Trigger repo
In Verification			Trigger repo					Trigger repo
Waiting for response								
StageRollout					Trigger repo	Do not trigger repo	Trigger repo	Trigger repo
UMDStore								
(terminal state)								
Deferred				Trigger repo				
Production								Trigger repo
(terminal state)								
Rejected								
(terminal state)								

**Table 1 RequestTracker progress states**

Additionally, the release version and the link to respective repository is stored within the ticket in the *ReleaseVersion* and *RepositoryURL*. Links to Quality Criteria Verification Report and Stage Rollout Report, which are stored in the EGI DocDB Document Server, are available through respective custom fields as well. Processing of the tickets within the *sw-rel* EGI RT queue is both automated and manual. If the ticket processing involves change its the *RolloutProgress* status then the EGI Repository is notified of this change. EGI Repository is especially provided with respective ticket id in order to allow the EGI Repository to interface with the EGI RT and further process the information stored within the ticket. The processing of the ticket in the EGI RT *sw-rel* queue involves the following major steps:

- On creation of the ticket the default values for custom fields are set. Also, the release metadata is validated again as it can be provided by the internal technology providers. If the

validation procedure is successful, then the *RolloutProgress* is switched automatically to the *Unverified* state and the members of Quality Criteria Verification group are notified automatically using the standard RT AdministrativeCc notification and the ticket moves to *InVerification* area.

- If the Quality Criteria Verification process passes successfully, then the ticket moves to the *StageRollout* area. Respective Stage Rollout group is notified automatically again. Move to the *StageRollout* area also involves automatic spawning of a child ticket in the *staged-rollout* queue where the stage rollout process is tested thoroughly. The child ticket contains the Repository URL, Release Version and Release Information and link to the Quality Criteria Verification report from the parent ticket.
- The tickets within the *staged-rollout* queue track the *Outcome* (*Accept* or *Reject*) of the stage rollout process. Moreover, a separate queue is used in order to allow finer grained notifications of the process especially towards the groups of Early Adopters. The ticket also provides a custom field to hold the Stage Rollout report. After the process is finished and the ticket in the *staged-rollout* queue is set as resolved, the Staged Rollout report link is transferred back to the parent ticket in the *sw-rel* queue.
- Once the stage rollout process successfully finishes and the workflow returns back to the *sw-rel* queue, respective ticket moves to the *Production* area and members of the Production group are notified automatically.

### 3.3.2.2 Integration with GGUS

Integration with GGUS provides the external technology providers with an entry point into the software provisioning workflow. The integration with GGUS is implemented as a RT – GGUS web services based interface.

The RT – GGUS interface consists of two parts, both based on the Perl SOAP::Lite[R 14] library. One part is the EGIRTforGGUS web service which provides the GGUS – RT interface on the RT side. The EGIRTforGGUS web service offers *TicketCreate*, *TicketModify* and *AddTicketAttachment* methods. *TicketCreate* is used to create new ticket in the *sa2-umd-rel* queue. A notable parameter of the *TicketCreate* method is the *GGUSTicketID* which is stored within respective custom field in the newly created RT ticket and used for further reference to the original GGUS ticket. The *TicketModify* method allows adding new correspondence to an existing RT ticket and/or changing its status. Finally, *AddTicketAttachment* is this particular case used to pass the ReleaseMetadata as GGUS ticket attachment to the respective custom field of a ticket in the RT *sa2-umd-rel* queue.

The other part of the interface consist of RT scrips invoked when a ticket in the *sa2-umd-rel* queue is modified or changed its status. These scrips call the *TicketModify* methods of the GGUS Grid HelpDesk web service. The GGUS Grid HelpDesk *TicketModify* method is used to add correspondence to the GGUS ticket diary of steps or change the GGUS ticket status. The method returns a matching *GGUSTicketID* of respective ticket in GGUS when its call was successful.

### 3.3.3 Repo-backend Component

The Repo-backend component is the heart of the business layer. It is responsible for sanitizing requests to the data layer and retaining the logical integrity of the data throughout operations. It interfaces directly to the data layer and transforms data requested from (or sent to) its API into the form appropriate for presentation (or storage accordingly). The Repo-backend component is divided into four modules:

- Repo daemon
- YUM/APT Repositories Generator
- Early-access Builder and
- Repo mirroring facility.

High-level details about the functionality offered by each module are provided in the next paragraphs.

### 3.3.3.1 Repo Daemon

Upon *RolloutProgress* change, the Repo daemon module receives notifications (via its HTTP API) originated from the RT component. It fetches the release details stored within the RT ticket as well as the associated release XML file and it recognizes whether the commanded *RolloutProgress* transition is an allowed one or not.

In case of a valid *RolloutProgress* transition, the Repo daemon is the responsible module for:

- parsing the given release XML file
- performing a logical validation of the content of the release XML file. Example cases that could cause the logical validation to fail are: a Major release that is defined as an incremental one, an emergency release that is stated as a non-incremental one, etc.
- validating the content of the given release XML file against the data exist in the EGI repository database. An indicative scenario, in which an error should be raised out if this validation, could be: a non-incremental minor release of a product has been submitted but the associated Major is not (yet) released into the production area
- storing the release metadata into the EGI database
- moving or downloading, in case of a new release submission, the corresponding bundle of artefacts (RPMs, DEBs, ...) into the designated EGI Repository area, in respect to the *RolloutProgress* value
- triggering the YUM/APT repositories generator module in order to create the corresponding YUM or APT repositories
- and finally, regardless the result of this process (success or failure), the Repo daemon is the responsible module for communicating the outcome back to the RT component.

Moreover, the EGI Repo daemon, offers a built-in RESTful API, for intercommunication with the rest of the EGI Repository components, providing, in this way, access to the EGI Repository pool of metadata (data layer). In addition, it offers an RSS based mechanism for integration with the WebPortal component and it provides an XML validation service as well as an XMLtoHTML conversion facility.

Finally, a web-application (<https://admin-repo.egi.eu/admin/>) is under development, which allows a group of authorized members of the SA2 activity to monitor the product releases *RolloutProgress* transitions, as those conducted by the RT component. Furthermore, the said web-application provides an overview of the versioned product releases distribution within the identified EGI Repository areas/states.

Queue										
Unverified										
Deferred										
StageRollout										
Production										
UMDStore										
Rejected										
EGI Repository: Queue										
Id	Ticket	Previous RP	Current RP	Processed	External	Status	Message Sent	Message	Inserted	Last Updated
263	<a href="#">1834</a>	In verification	StageRollout	yes	yes	Success	yes	<a href="#">show...</a>	2011-05-04 10:36:10	2011-05-04 10:36:04
262	<a href="#">1853</a>	Submitted	Unverified	yes	yes	Success	yes	<a href="#">show...</a>	2011-05-04 09:58:04	2011-05-04 09:56:48
261	<a href="#">1852</a>	Submitted	Unverified	yes	yes	Success	yes	<a href="#">show...</a>	2011-05-04 09:57:59	2011-05-04 09:56:36
260	<a href="#">1851</a>	Submitted	Unverified	yes	yes	Success	yes	<a href="#">show...</a>	2011-05-04 09:57:55	2011-05-04 09:56:18
259	<a href="#">1850</a>	Submitted	Unverified	yes	yes	Success	yes	<a href="#">show...</a>	2011-05-04 09:57:50	2011-05-04 09:56:05
258	<a href="#">1849</a>	Submitted	Unverified	yes	yes	Success	yes	<a href="#">show...</a>	2011-05-04 09:57:45	2011-05-04 09:55:50
257	<a href="#">1848</a>	Submitted	Unverified	yes	yes	Success	yes	<a href="#">show...</a>	2011-05-04 09:57:41	2011-05-04 09:54:53
256	<a href="#">1847</a>	Submitted	Unverified	yes	yes	Success	yes	<a href="#">show...</a>	2011-05-04 09:57:36	2011-05-04 09:54:35
255	<a href="#">1846</a>	Submitted	Unverified	yes	yes	Success	yes	<a href="#">show...</a>	2011-05-04 09:57:31	2011-05-04 09:54:05
254	<a href="#">1845</a>	Submitted	Unverified	yes	yes	Success	yes	<a href="#">show...</a>	2011-05-04 09:57:27	2011-05-04 09:53:48
253	<a href="#">1844</a>	Submitted	Unverified	yes	yes	Success	yes	<a href="#">show...</a>	2011-05-04 09:57:22	2011-05-04 09:53:29
252	<a href="#">1843</a>	Submitted	Unverified	yes	yes	Success	yes	<a href="#">show...</a>	2011-05-04 09:57:17	2011-05-04 09:53:17
251	<a href="#">1842</a>	Submitted	Unverified	yes	yes	Success	yes	<a href="#">show...</a>	2011-05-04 09:57:13	2011-05-04 09:53:00
250	<a href="#">1841</a>	Submitted	Unverified	yes	yes	Success	yes	<a href="#">show...</a>	2011-05-04 09:57:08	2011-05-04 09:52:55
249	<a href="#">1840</a>	Submitted	Unverified	yes	yes	Success	yes	<a href="#">show...</a>	2011-05-04 09:57:03	2011-05-04 09:51:08

Figure 6 Backend Component Admin Interface

### 3.3.3.2 YUM/APT Repositories Generator

As the name implies, the main aim of the module is to create the YUM and/or APT related repositories based on a specific set of attributes indicated by the technology provider. Furthermore, the module is following an asynchronous implementation logic, primarily for :

- guaranteeing a FIFO sequence for processing the requests as they are commanded by the different components and modules of the EGI Repository setup.



- to avoid causing delays to the overall workflow
- to be easy replicated to different hosts as the creation of YUM repositories can be realized only in a RPM-based OSs and for the construction of APT repositories a DEB-like OS might be needed.

The most important internal functions carried out by the module are:

- Provides an interface for accepting and queuing YUM and/or APT repositories creation commands originated by the other internal modules or components included in the business layer of the EGI Repository structure (for the time being it accepts commands originated by the Repo daemon, the Early-access builder and the UMD Composer).
- Communicates with the Repo DB in order to fetch the metadata needed for the YUM and/or APT repositories creation
- Builds the YUM and/or APT repositories, in respect to the accepted command.
- Generates, the YUM and/or APT related repofiles.
- Updates the Repo database upon completion.

### 3.3.3.3 Early-access Builder

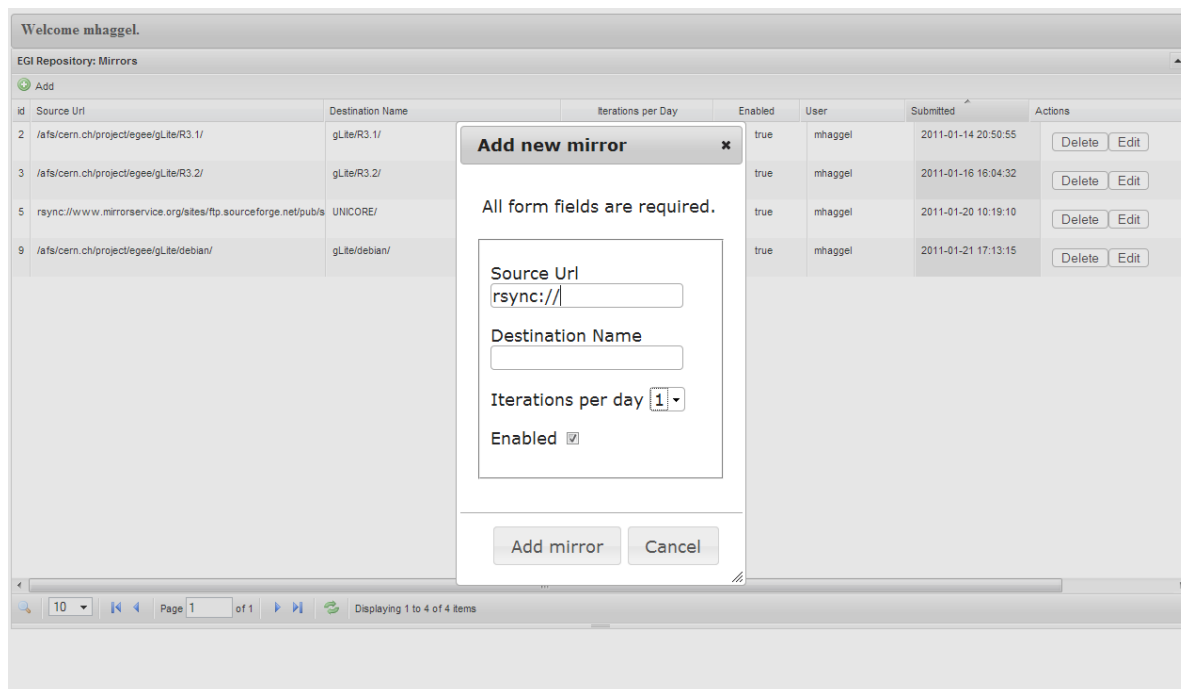
As it has been mentioned in paragraph 3.2.1, “early-access” is the area that holds the bundle of the artefacts, associated with the versioned product releases which have been, successfully, passed into the Stage-Rollout phase, they are targeting a specific UMD-Major release, but they are not yet released into the production area. The Early-access Builder is the responsible module for building and keeping this area up-to-date. In detail, it communicates periodically with the repository database and upon new insertion (or removal) of a product release into (from) the “UMDStore” area it performs the following actions:

- (re)builds the “early-access” area
- triggers the YUM/APT Repositories Generator module in order to (re)create the associated YUM and/or APT repositories

### 3.3.3.4 Repo Mirroring Facility

The Repo mirroring facility module is an autonomous sub-system, independent from the EGI Software provision workflow that aims to provide mirroring capabilities to a group of authorized members of the SA2 activity (available at <https://admin-repo.egi.eu/admin/> using the Mirrors button). By using the module’s dedicated web interface, the user is able to set (or unset) periodic mirroring jobs, replicating and therefore make available through the EGI Repository any remote rsync site or service. Moreover, any folder existing in the mounted CERN’s AFS file system could be replicated through this module.





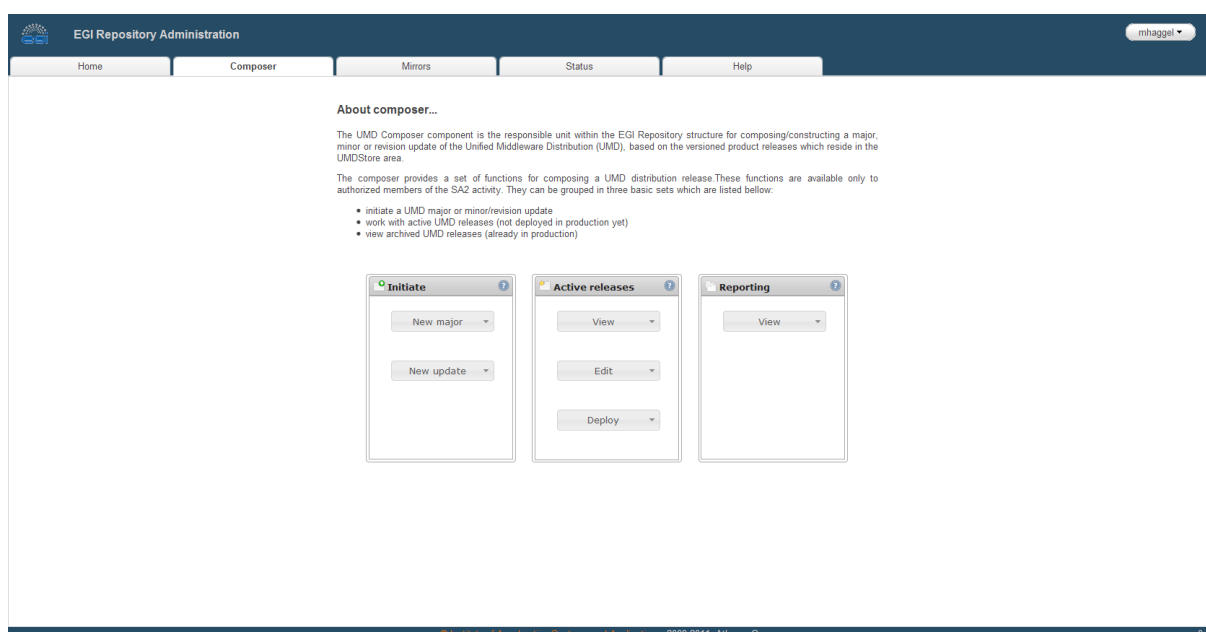
**Figure 7 Repository Mirroring facility Admin interface**

### 3.3.4 UMD Composer

The *UMD Composer* component is the responsible unit within the EGI Repository structure for composing/constructing a major, minor or revision update of the Unified Middleware Distribution (UMD), based on the versioned product releases which reside in the “UMDStore” area. The said versioned product releases are of a specific platform and architecture pair and further on this document they will be referred as PPAs.

Through the component’s user interface (currently under development), three key functionalities are available to a group of authorized members of the SA2 activity. They can:

- initiate a UMD major or minor/revision update,
- work with active UMD releases (not deployed in production yet) and
- view archived UMD releases (already in production).



**Figure 8 UMD Composer - key functions offered (main page).**

The “initiation” function offers the capability of starting/initiating a UMD major or minor/revision release to work with. By starting a new major release, the user is informed by the system with the next available major number and is prompted to select the appropriate triplet, defining in this way the platform(s) (OS), the architecture(s) (ARCH) and the automated package installation technology (YUM/APT) that will be used for the newly initiated release.

Moreover, acting upon initiation of a new update for a given major, the system prompts the user to select the type of the update, in terms of whether it is a minor or a revision one. The system informs the user of the next possible minor or revision number, respectively, and prompts to accept or reject the “initiation”. Further, two important rules constrain the update initiation process:

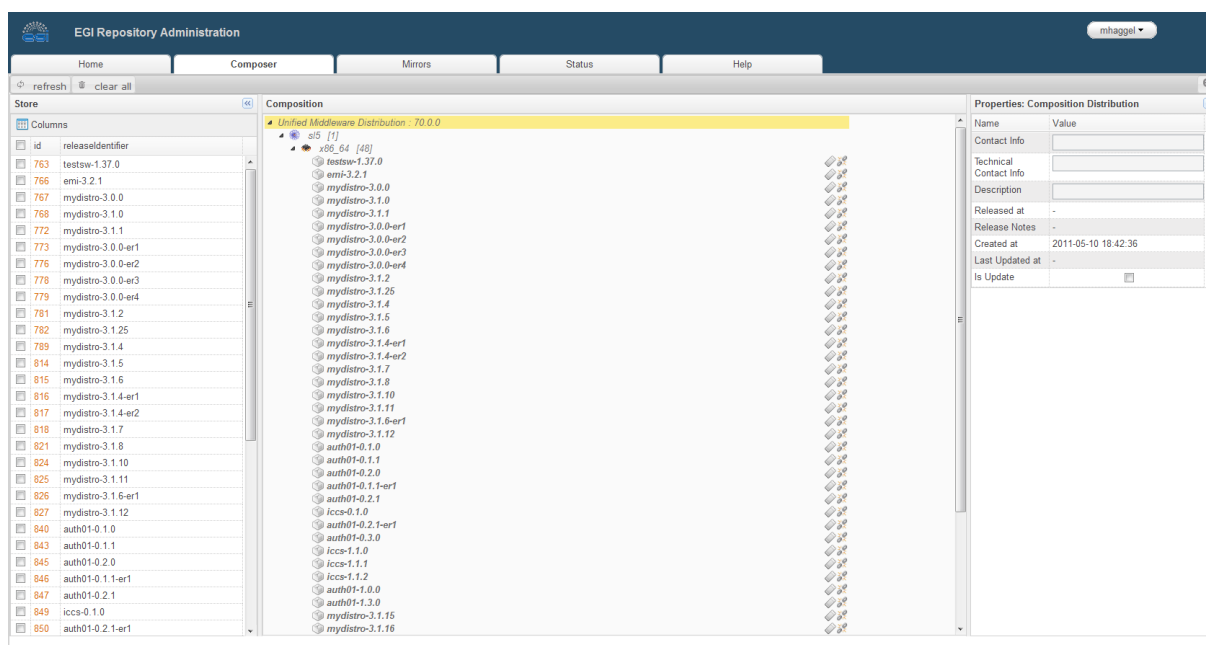
- Only one update (minor or revision) per respective major version may be initiated and therefore be under the composition workflow.
- A revision version of a given UMD minor release may not be initiated while a minor with a greater number is under composition, or has been deployed in production: For example, release 4.4.2 cannot be initiated if the 4.5.0 is under composition or has been already deployed to production.

The next key function offered to the user by the *UMD Composer* component is the capability to work with the initiated UMD releases that are not deployed yet in the production. Under this scope, view, edit/modify and deployment actions can be performed.

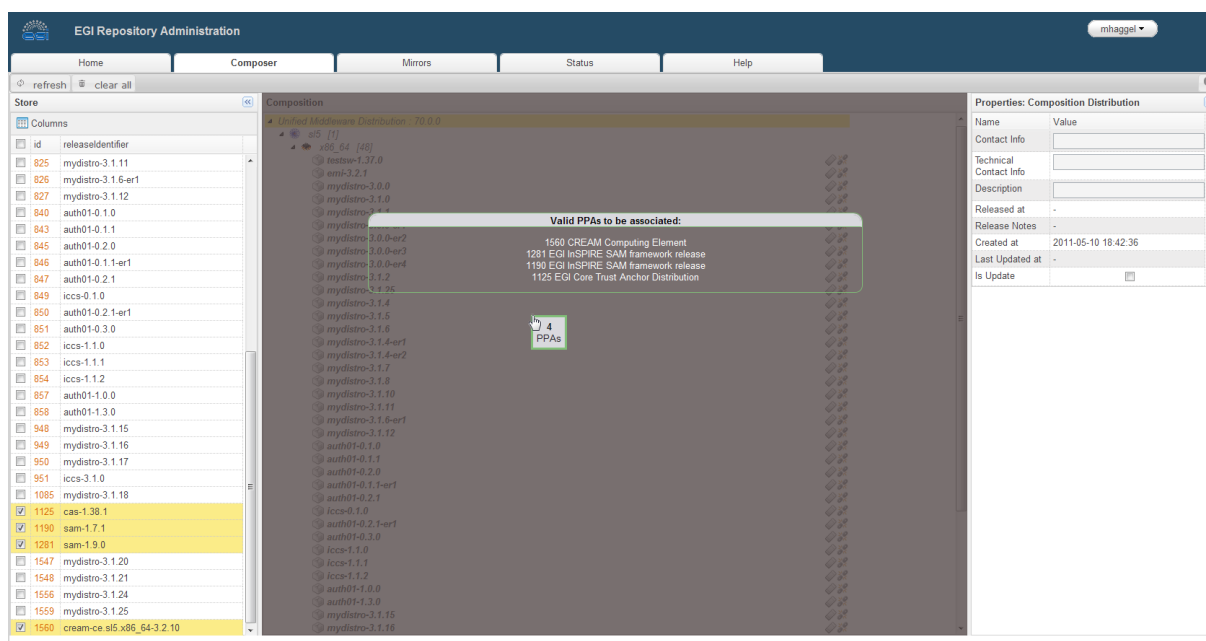
Viewing an active release could be performed by any authenticated user, regardless his authorization

level. Through this action, one is able to review the PPAs that are already associated to a given active UMD release, access the metadata of each PPA (associated or not) and evaluate the active UMD release metadata.

Editing/modifying an active UMD release is permitted only by specific members of the SA2 activity, granted with the analogous authorization level. During this action, one is able to modify an active UMD release by adding or removing associated PPAs and/or alter the related to the release, metadata (i.e. release notes, changelog, documentation links, known issues etc.).



**Figure 9 UMD Composer, the composition workspace – Default view**



**Figure 10 UMD Composer's composition workspace - Drag & Drop effect, instant preview of the PPA – UMD release associations being performed.**

Deploying a UMD release into production is performed through the UMD Composer component, as well. Alike editing related actions, it is permitted only to specific members of the SA2 activity, granted with the analogous authorization level. Upon successful UMD release deployment, the component communicates with the Repo daemon and updates EGI Repository database by setting the state of each associated PPA to “production”. This change is further communicated to the RT component by the Repo daemon.

The last, but not least, key function of the *UMD Composer* provides the user with a review and reporting infrastructure for archived UMD releases that are deployed in production. However, due to the limited timeframe that is available for the design, development and testing of the component, this functionality will not be ready on the first release of the *Composer* scheduled for June.

### 3.4 External interaction layer

#### 3.4.1 Web Portal

The portal for EGI Software Repository is deployed as a Wordpress site [R 9], available at <http://repository.egi.eu>. The site shares the same look and feel as the official EGI site [R 10], thanks to a custom child theme [R 11] that was developed based on Wordpress' default theme (Twenty Ten). General information about the repository and the EGI initiative are provided, such as the goals of the project, user support contact points, etc. In addition, the portal serves as an entry point, providing a user interface where users can browse through the releases of popular grid middleware software that have reached the *StageRollout* and *Production* stages of the EGI workflow as well as preliminary

installation guides. In specific, the information supplied by Technology Providers and displayed for each release is as follows, a short description for the software release, links to further documentation, the release notes and the link to the Repository URL where the users can download the aforementioned release.

At the time being the EGI Software Repository portal is automatically updated with the new releases that make it into the Repository through the EGI workflow. The communication between the Repository and the portal is achieved through RSS feeds. The repository maintains an RSS feed channel, adding an RSS feed for each new release it processes. The portal retrieves the feeds and translates each one of them into a blog post containing basic information about the release as described above. This procedure is implemented by two Wordpress plugins; the first one is responsible for acquiring any new feeds in the channel and transforming them into blog posts [R 12], whereas the second one was designed to work as a filter that customizes the information included in the posts, by extracting custom fields of the feed introduced especially for the communication between the portal and the repository.

In the future we plan to extend the RSS functionality in order to include new UMD releases in the portal. A browsing interface will be developed inside the portal enabling users to browse UMD releases and the available packages per platform for the different products of each release.

### 3.4.2 Repository Provision Site

The repository provision site offers access to the EGI 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 and architectures. Currently, the EGI Repository structure consists of two instances of the repository provision site, on a DNS round-robin mechanism, to ensure high-availability. The main areas that are populated by this site and they are of significant importance for the stability of the overall software provisioning workflow, could be accessed at the following locations:

- Location: <http://admin-repo.egi.eu/sw/unverified/>  
Provide access to the YUM/APT repositories associated with the products releases that are under the 'quality verification process'. These repositories are acting as automated package installation, for the sites that are participating into the said process.
- Location: <http://repository.egi.eu/sw/stagerollout/>  
Provide access to the YUM/APT repositories associated with the products releases that are under 'stage-rollout process'. These repositories are acting as automated package installation, for the sites that are participating into the said process.
- Location: <http://repository.egi.eu/sw/production/>  
Provide access to the YUM/APT repositories associated with the products that have reached the production level. These repositories are acting as automated package installation, for every site within the entire EGI infrastructure.






- Location: <http://repository.egi.eu/sw/early-access/>  
This is a dedicated area for the UMD distribution. It provides access to the YUM/APT repositories associated with the products that are targeting a specific UMD Major release but they are not yet included into the production-level UMD snapshot.

### 3.4.3 GGUS Component and Dashboard

The Global Grid User Support (GGUS) system is the primary means by which users request support when they are using the grid. As such the GGUS provides an entry point to the EGI software provisioning workflow for the external technology providers such as European Middleware Initiative (EMI). For the EGI software provisioning workflow the GGUS implements a software release ticket submission form, software release dashboard and an interface to the EGI RT where the software provisioning workflow is actually undertaken.

The software release ticket submission form allows the external technology providers to upload the release metadata for the software being released. Upon this action GGUS interfaces with the EGI RT and creates a new ticket as described in section 3.3.2.2. Changes to the tickets in the EGI RT are communicated back to the GGUS. The external technology providers can then review the changes using the software release dashboard which provides an overview of existing tickets and their status.

**►Technology Helpdesk**




[►Software Release Dashboard](#)
[►Technology Support Dashboard](#)
[►Go to GGUS](#)

**Open Software Release Ticket**

User information			
Name	Milos Liska ( - Release Manager)	E-Mail	xliska@fi.muni.cz
Cc to	<input type="text"/>	Notification mode	<input type="radio"/> on every change <input checked="" type="radio"/> on solution

Release information	
Current Date / Time	2011-05-10 05:37 UTC
Release title (required)	<input type="text"/>
Details (optional)	<div style="border: 1px solid black; height: 60px;"></div>
Attach release description (must be named: "release.xml")	<input type="button" value="Choose File"/> No file chosen
<input type="button" value="Submit"/>	

**Figure 11 EGI Software Provisioning Submission Form**

►Technology Helpdesk

►Go to GGUS   ►Open Software Release Ticket

### Software Release Dashboard

**Software Release Tickets**   Page loaded: 2011-05-10 05:34 UTC

Show  tickets, priority:  status:  Support unit:  ordered

13 tickets found.

ID	Status	Priority	Date of problem	Support Unit	Info
► 46508	assigned	less urgent	2011-05-04 08:08	EGI Software Provisioning	2011-05-04 08:08 UTC
► 46507	assigned	less urgent	2011-05-04 07:32	EGI Software Provisioning	new test
► 46506	assigned	less urgent	2011-05-04 06:44	EGI Software Provisioning	DryRun full 2nd try
► 46505	assigned	less urgent	2011-05-04 06:38	EGI Software Provisioning	DryRun full
► 46499	assigned	less urgent	2011-04-28 08:09	EGI Software Provisioning	status test
► 46490	assigned	less urgent	2011-04-19 11:28	EGI Software Provisioning	Test for revoking tickets
► 46480	assigned	less urgent	2011-04-12 06:18	EGI Software Provisioning	test tuesday
► 46478	assigned	less urgent	2011-04-11 11:41	EGI Software Provisioning	test test
► 46477	assigned	less urgent	2011-04-11 11:32	EGI Software Provisioning	test ige release
► 46476	assigned	less urgent	2011-04-11 11:31	EGI Software Provisioning	torsten
► 46446	assigned	less urgent	2011-03-29 14:43	EGI Software Provisioning	Test xyz
► 46400	assigned	less urgent	2011-03-25 12:41	EGI Software Provisioning	My test
► 46399	assigned	urgent	2011-03-25 10:04	EGI Software Provisioning	test release

► Contact the GGUS team  
► Go to GGUS

**Figure 12 EGI Software Provisioning Dashboard**

## 3.5 Auxiliary Components

### 3.5.1 Replication Mechanism

High-availability of the EGI repository service (<http://repository.egi.eu>) is implemented using two web servers replicating the repository data and the web portal information. The replication procedure is completely automated for both services. Additionally, load balancing between the web servers is implemented using round-robin DNS<sup>4</sup>.

Repository data (<http://repository.egi.eu/sw/> and <http://repository.egi.eu/mirrors/>) are replicated using a cron<sup>5</sup>based synchronization algorithm. Each server is responsible of running the replication procedure, keeping the package information up to date.

The web portal is designed using Wordpress, which is an open source publishing platform powered by PHP and MySQL. The Wordpress synchronization procedure is divided into two parts: The synchronization of the database, and the file structure. The database synchronization uses MySQL's

<sup>4</sup>Round Robin DNS: [http://en.wikipedia.org/wiki/Round-robin\\_DNS](http://en.wikipedia.org/wiki/Round-robin_DNS)

<sup>5</sup>Cron is a time-based job scheduler in Unix-like computer operating systems.



Master-Slave replication feature [R 15], enabling data transfer from one MySQL database server (the master) to be automatically replicated to one or more MySQL database servers (the slaves). Replication is asynchronous - slaves need not to be connected permanently to receive updates from the master. This means that updates can occur over long-distance connections. Configuration allows replicating complete databases, schemas, or even only a single database table. In order to prohibit a stale data condition a cron job replicates every 5 minutes the contents of the slaves from the master.

The replication of the Wordpress file structure is achieved using an SVN repository service (<https://svn.hellasgrid.gr/svn/repository.egi.eu/>) deployed on each load-balanced web server, which keep local copies of the Wordpress file structure in synch.

Wordpress contents are administrated and created using a dedicated, separate server, <http://wp-admin.repository.egi.eu>. This server holds the master MySQL database, and has write permission to the repository (as opposed to the replicating, load-balancing slave servers. New posts will be inserted using this server and the changes will be automatically deployed to the repository and slave databases. Administrative tasks like, plug-in update, can be performed in a local checkout. Changes are committed to the repository automatically update the Wordpress administration server and the three web servers.

### 3.5.2 Metrics Collection

A seamless procedure is designed and deployed in order to collect, analyse and display repository and web portal metrics. The web server logging data are collected in a single server using syslog-ng [R 16]. The syslog-ng application supports reliable and encrypted transport, and offers powerful message filtering, sorting, pre-processing and log normalization capabilities. Utilizing message parsing and classification, syslog-ng is able to correlate log messages.

The collected logging data are processed and represented by AWStats [R 17]. AWStats is a freetool that generates advanced web, streaming, ftp or mail server statistics, graphically. This log analyzer works as a CGI to a web server or from command line, and shows all possible information that log file contains, in graphical web pages. AWStats is free software distributed under the GNU General Public License. As AWStats works from the command line but also as a CGI, it can work with all web-hosting providers, which allow Perl, CGI and log access.



## 4 NEXT STEPS – PLANS

As we are currently working in full speed to finalise the implementation of the 2nd iteration of the EGI Repository in time for the 1st release from the EMI Project [R 18] there are no concrete plans for the 2nd year of the project. Therefore the main focus for the 2<sup>nd</sup> year is to optimise the current workflow as needed based on the results/feedback we will get from the 1<sup>st</sup> releases of UMD. We plan however to design and implement most of the following.

- Enhance the *UMD Store* functionality to add some kind of archiving/reporting feature of the *UMD Composer*.
- Provide browsing facilities per capability, and list available products including a short description of each.
- Provide RSS feeds for new releases to the end-users/sites.
- Provide support for Debian based platforms.
- Optimise the workflow based on the experience running it in full scale.
- Adhere to the requests/needs of the communities involved.
- Prepare for prospective migration to Cloud computing.

## 5 REFERENCES

R 1	MS501 - Establishment of the EGI Software Repository and associated support tools <a href="https://documents.egi.eu/document/46">https://documents.egi.eu/document/46</a>
R 2	MS504: EGI Software Repository Architecture And Plans <a href="https://documents.egi.eu/document/89">https://documents.egi.eu/document/89</a>
R 3	MS402 - Deploying software into the EGI production infrastructure <a href="https://documents.egi.eu/document/53">https://documents.egi.eu/document/53</a>
R 4	The Repo view: <a href="http://repository.egi.eu/sw">http://repository.egi.eu/sw</a>
R 5	The Portal of the Repository: <a href="http://repository.egi.eu/">http://repository.egi.eu/</a>
R 6	UMDRoadmap <a href="http://go.egi.eu/UMDRoadmap">http://go.egi.eu/UMDRoadmap</a>
R 7	MS503: Software Provisioning Process <a href="https://documents.egi.eu/document/68">https://documents.egi.eu/document/68</a>
R 8	EGI Glossary <a href="https://wiki.egi.eu/wiki/EGI_SA2_Glossary">https://wiki.egi.eu/wiki/EGI_SA2_Glossary</a>
R 9	<a href="http://wordpress.org/">http://wordpress.org/</a>
R 10	<a href="http://www.egi.eu">http://www.egi.eu</a>
R 11	<a href="http://codex.wordpress.org/Child_Themes">http://codex.wordpress.org/Child_Themes</a>
R 12	<a href="http://feedwordpress.radgeek.com/">http://feedwordpress.radgeek.com/</a>
R 13	Resource Tracker (RT) : <a href="http://bestpractical.com/rt/">http://bestpractical.com/rt/</a>
R 14	SOAP lite: <a href="http://www.soaplite.com/">http://www.soaplite.com/</a>
R 15	MySQL Master Slave Replication feature: <a href="http://dev.mysql.com/doc/refman/5.0/en/replication.html">http://dev.mysql.com/doc/refman/5.0/en/replication.html</a>
R 16	Syslog-NG: <a href="http://en.wikipedia.org/wiki/Syslog-ng">http://en.wikipedia.org/wiki/Syslog-ng</a>
R 17	Awstats: <a href="http://awstats.sourceforge.net/">http://awstats.sourceforge.net/</a>
R 18	European Middleware initiative (EMI): <a href="http://www.eu-emi.eu">http://www.eu-emi.eu</a>
R19	<a href="https://documents.egi.eu/public/ShowDocument?docid=428">https://documents.egi.eu/public/ShowDocument?docid=428</a>
R 20	Guide For The Internal Technology Providers: <a href="https://documents.egi.eu/document/428">https://documents.egi.eu/document/428</a>
R 21	Release Xml For External Technology Providers: <a href="https://documents.egi.eu/document/399">https://documents.egi.eu/document/399</a>

## 5.1 Table of Figures

Figure 1 Internal Technology Providers Provisioning Process.....	10
Figure 2 External Technology Providers Software Provisioning Process .....	11
<b>Figure 3: Internal Technology Provider Release example.....</b>	<b>13</b>
Figure 4: External Technology Provider Release example. ....	14
Figure 5: EGI Repository architecture diagram .....	15
Figure 6 Backend Component Admin Interface .....	23
Figure 7 Repository Mirroring facility Admin interface.....	25
Figure 8 <i>UMD Composer</i> - key functions offered (main page).....	26
Figure 9 <i>UMD Composer</i> , the composition workspace – Default view.....	27
Figure 10 <i>UMD Composer's</i> composition workspace - Drag & Drop effect, instant preview of the PPA – UMD release associations being performed. ....	28
Figure 11 EGI Software Provisioning Submission Form.....	31
Figure 12 EGI Software Provisioning Dashboard.....	32
Figure 13 Entity Relationship Diagram of the Metadata Schema .....	37

