EUROPEAN MIDDLEWARE INITIATIVE

SOFTWARE RELEASE PLAN

EU DELIVERABLE: DSA1.2

Document identifier:	**EMI-DSA1.2-SoftwareReleasePlan-v0_2.odt**
Date:	**07/01/2010**
Activity:	
Lead Partner:	
Document status:	
Document link:	

Abstract:
This document describes the release procedures and policies for the middleware services and the complete EMI reference distributions. It also contains the initial release schedule to be prepared in collaboration with the PTB and the JRA1 Work Package. The release schedule will be updated every three months during the course of the project

Delivery Slip

	Name	Partner / Activity	Date	Signature

From	Cristina Aiftimiei	INFN/SA1	01/07/10	
Reviewed by				
Approved by				

Document Log

Issue	Date	Comment	Author / Partner
1	01/07/2010	Table of Contents	Cristina Aiftimiei
2	15/07/2010	Added Chapter 3	Cristina Aiftimiei
3	24/10/2010	Added Release Categories, Release Criteria	Cristina Aiftimiei

Document Change Record

Issue	Item	Reason for Change
1		
2		
3		

TABLE OF CONTENTS

1. INTRODUCTION

1.1. PURPOSE

<This section contains a description of the purpose of the document in the context of the EMI project>

1.2. DOCUMENT ORGANISATION

<This section contains a description of how the document is organized and a very brief description of each chapter>

1.3. REFERENCES

R1	
R2	
R3	
R4	
R5	
R6	

DOCUMENT AMENDMENT PROCEDURE

This document can be amended by *<person or team that can make changes to this document>* further to any feedback from other teams or people. Minor changes, such as spelling corrections, content formatting or minor text reorganisation not affecting the content and meaning of the document can be applied by the *<person or team that can make changes to this document>* without peer review. Other changes must be submitted to peer review and to the EMI PEB for approval.

When the document is modified for any reason, its version number shall be incremented accordingly. The document version number shall follow the standard EMI conventions for document versioning. The document shall be maintained in the CERN CDS repository and be made accessible through the OpenAIRE portal.

1.4. TERMINOLOGY

<XYZ>	<Description and references if needed>

2. EXECUTIVE SUMMARY

<High-level summary of the content of this document. It should provide the reader with information about the structure of the document the most important elements of its content. Max one page.>

3. SUMMARY OF INITIAL SITUATION

This section presents a short overview of the release processes of the 4 MW providers, ARC, dCache, gLite, UNICORE, that constitute the basis on which the EMI Release process will be developed.

3.1. ARC RELEASE PROCESS

The **Advanced Resource Connector (ARC)** middleware, developed by NorduGrid collaboration, is a software solution that uses Grid technologies to enable sharing and federation of computing and storage resources distributed across different administrative and application domains. ARC is used to create Grid infrastructures of various scope and complexity, from campus to national Grids.

This is a summary of the information present at [1] (http://wiki.nordugrid.org/index.php/Release_management).

3.1.1 Background

An ***ARC release*** is defined as a set of source packages AND the corresponding binaries packages on the supported platforms. The source release is defined by an SVN tag. *ARC releases* have been created with the strong emphasis on multiplatform support, Linux distribution independence and recently also extended OS independence (Linux, Windows, Mac, Solaris), non-intrusiveness and ease of operation

3.1.2 Development Environment

- code is kept in svn spread over several svn trees, ARC has a non-monolitic source code tree (svn.nordugrid.org)
- Bugzilla is used to report software defects and to act upon them; feature requests are also collected in Bugzilla; bugs, including feature requests are periodically reviewed and prioritized
- WIKI [ref http://wiki.nordugrid.org/] is used as a workarea: to collect development plans, technical drafts, release coordination, configuration templates, quick guides, HOWTOs
- nordugrid-discuss mailing: high traffic technical list
- Ticketing system: an issue tracking system is used to provide user support
- Build and testing systems: currently there are two production systems in use to perform automatic nightly builds of several svn areas and to produce binary packages for tagged releases

3.1.3 Release Categories

Major release: longer term planning, 3-6 months preparation, alpha, beta, rcx test releases, introduces new components, featuresobsoletes components, may break backward compatibility

Minor release: planned bugfix release, max one month preparation, no new features, no new components, only fixes, scheduled release

Emergency release: unplanned urgent release to fix a security issue or a critical bug, maximum two weeks preparation

Binary update of a release: sometimes due to a change in the external dependencies ARC releases have to be rebuilt.. This does not affect the ARC source code, no re-taging takes place, only new binaries are buil and it can happen for both major, minor or emergency releases

3.1.4 Workflow

3.1.4.1 Scheduled release procedure:

- Major or minor releases follow a scheduled release procedure: The contents of major ARC releases are trying to follow a high level ARC development roadmap
- Planning phase (community process): prioritizing open bugs to be fixed; collecting requests for new features (only for major release);
- Decision phase: decision takes into account the status (maturity) of release candidate components, the release content is defined in terms of components and bugs to be fixed; supported platforms decided
- Preparation of the release: creation of the release branch, documentation updates, release notes, release candidate(s) are created from the branch
- Testing: Release candidate is deployed on the test infrastructure where the test suits are executed
- Release is ready if: All the critical bugs recorded in the bugzilla are closed; functionality decided in planning phase is successfully tested on the release candidate; Binary packages are available for all supported platforms; Documentation reflecting changes from the previous release is in place

3.1.4.2 Emergency Release Procedure:

- Applied when running into a MAJOR ISSUE: a software defect considered critical or a security whole was discovered
- the reporting of the MAJOR ISSUE can happen through several channels: bugzilla, mailing list, direct contact to core coordination group members
- The emergence release is being prepared:
- Emergency release is ready, the process can't take more time than two weeks

3.2. DCACHE RELEASE PROCESS

dCache a system for storing and retrieving huge amounts of data, distributed among a large number of heterogenous server nodes, under a single virtual filesystem tree with a variety of standard access methods. Depending on the Persistency Model, dCache provides methods for exchanging data with backend (tertiary) Storage Systems as well as space management, pool attraction, dataset replication, hot spot determination and recovery from disk or node failures.

3.2.1 Background

Time Based Releases: dCache is moving away from feature based releases towards Time Based Releases. As the name already implies, this essentially means that it will be better in predicting when a release is being published. The drawback is that features which don't make it into the repository till some time before the release is due, just won't be part of that release. The advantage is that sites can do better planning in upgrading and that the synchronization with distributors is eased.

3.2.2 Development Environment

- for version control code is kept in SVN: (http://svn.dcache.org/dCache/)
- RT is used to keep track of software defects and to act upon them; feature requests are also collected in RT bugs [reference http://rt.dcache.org/Ticket/Display.html]

- WIKI (reference)http://trac.dcache.org/projects/dcache/wiki] is used as a workarea: to collect development plans, technical meetings, release coordination, documentation links
- Build and testing systems: currently the Hudson build&test system [reference http://svn.dcache.org/build/] is used to perform automatic and on-demand builds and to produce binary (rpm) and source (tgz, pkg) packages for tagged releases

3.2.3 Release Categories

dCache follows a two dimensional release system. Every couple of months it is produced a feature release. Feature releases have version numbers 1.9.x-1, where x is a positive integer. Each feature release is followed up by a number of maintenance releases containing bug fixes or high priority feature tweaks. Maintenance releases for release 1.9.x-1 have version numbers 1.9.x-y, for y being an integer larger than 1.

- **Recommended Releases:** At any point in time, dCache.org will recommend a particular release that should be used on production systems. Other releases are considered stable, however care should be taken as such releases have only received limited amount of public testing or are outdated.
- **Golden Release:** One release is declared a “golden release” that means it is supported for a long period, in the order of a year or more. For golden releases there are strict rules concerning the patches which are allowed. (e.g. no new features)

3.2.4 Workflow

Two weeks before the release is declared to be cut, the repository is closed and no check-in is allowd anymore. At the same time, start testing.

Once a developer has done Unit Testing the code will be submitted to **Review Board**. Here the other developers can and should have a look at this code. Code has to be reviewed by at least one of the other developers and can be committed to the trunk branch as soon as all the developers who reviewed it mark it to be shipped. When code is committed commit hooks automatically build dCache, dcap and srm client. On error an email is sent to the developer who made the last commit. With the last successful build the yum repositories are updated and the rpms automatically installed on some machines.

If the code is intended to go into production branch the developer has to send a merge request to the Release Manager. Only the Release Manager can submit code to stable branches. A stable branch is any SVN branch from which releases are made.

3.3. GLITE RELEASE PROCESS

gLite is the next generation middleware for grid computing. Born from the collaborative efforts of more than 80 people in 12 different academic and industrial research centers as part of the EGEE Project, gLite provides a framework for building grid applications tapping into the power of distributed computing and storage resources across the Internet.

3.3.1 Background

[reference https://edms.cern.ch/document/973115]

gLite uses a *'continuous release process*' which was defined and optimized during EGEE-II and EGEE-III projects. The guiding principle of this approach is to manage independent updates in isolation from one another, allowing each to progress through the full software life-cycle without interference. This has proven to bring numerous benefits, particularly in accelerating the adoption of

important changes which can be properly prioritized and which are not held back by other unrelated updates. In this model there is a baseline release to which updates are continuously added.

3.3.2 Development Environment

[reference https://twiki.cern.ch/twiki/bin/view/EGEE/DevelopersGuide]

- code is kept in CVS. The gLite CVS repository is hosted by the CERN Central CVS Service [reference]
- Savannah [reference] is used to report software defects, to act upon them, and as release tracking tool;
- WIKI [reference https://twiki.cern.ch/twiki/bin/view/EGEE/EGEEgLite] is used as a workarea: to collect development plans, technical drafts, release coordination, configuration templates, quick guides, HOWTOs
- mailing-lists: project-eu-egee-midlleware-emt is used for developers discussions and announcements; weekly EMT [reference] telephone conferences are held for discussions on gLite Release Schedule, assess bug priorities.
- Ticketing system: GGUS ticketing system is used to provide user support [reference]
- Build and testing system: ETICS (eInfrastructure for Testing, Integration and Configuration of Software) is used to manage every aspect of the software development workflow, i.e. build, release and testing [ref Etics documentation]

3.3.3 Release Categories

[reference https://edms.cern.ch/document/1041827/1]

(to add in Terminology or reference on PT)

A PT will produce two different types of releases: **internal** and **production**. A *production release* is an update to a n*ode-type* (definition ??) and is considered production ready. An i*nternal release* is an update to individual, certified components that are made available to other product teams. The word 'internal' here denotes being internal to gLite, rather than internal to the Product Team.

Internal Releases – are initialized via a traditional Savannah patch. However, instead of being released to production, it will instead serve as input to node-type production patches. A modification will be required to Savannah to ensure that such patches are identifiable.

Production Releases - is an update to a self-contained product, typically node-type based, that will be deployed onto the production infrastructure. The product will typically contain a number of components that have been released internally by other product teams, however, it is the product team releasing the complete product to production that is ultimately responsible and accountable for that release.

3.3.4 Workflow

An overall description of the new gLite release process can be found in the EGEE-III gLite Product Team Release Process [reference] document. The process is based on product teams being entirely responsible for their products (development, testing, integration and release). The Product Team Release Process [reference https://twiki.cern.ch/twiki/bin/view/EGEE/ProdInt] consists on:

- Creating Patches and Metapackages: - PTs create in Savannah [ref] *metapackage patches*, if there are new metapackage versions, or i*nternal patches*, if there are new package versions that are not part of any PT metapackage but that are needed by other PTs

- Certification and Tests: - are under the PTs responsibility, following instructions on "How to certify a patch within a Product Team" guide [ref]. PTs should fill a certification report [ref https://twiki.cern.ch/twiki/bin/view/Main/CertReportTemplate]; beta-repositories are created
- Verification: tests reports are verified and all the basic tests are run.
- Staged roll-out: Only for metapackage patches Staged-Rollout repositories are created and SR-sites (EA – definition in glossary?)[ref] are announced. SR-sites write reports on results of their tests
- Production: Only for metapackage patches. The packages are copied from the beta repositories directly to production. Release notes must be prepared for every new release of a gLite product. The patches are released as part of an `Update`. An `Update` is a group of patches released in Production.

3.4. UNICORE RELEASE PROCESS

UNICORE (Uniform Interface to Computing Resources) offers a ready-to-run Grid system including client and server software. UNICORE makes distributed computing and data resources available in a seamless and secure way in intranets and the internet.

3.4.1 Background

UNICORE in EMI is a set of Client (UCC, HiLA) and Server (core, UVOS) packages, all available for Windows, Linux/UNIX and MacOS. The currently supported versions are: UCC 6.3.1, released on 07/07/2010; HiLA 2.0.0, released on 21/04/2010; core-server 6.3.1, released on 27/04/2010; UVOS 1.3.3, released on 30/03/2010

3.4.2 Development Environment

[reference TD]

- code is kept in SVN, hosted by SourceForge [reference http://unicore.svn.sourceforge.net/viewvc/unicore], with a mailing-list (*unicore-cvs-commits*) receiving notification on each commit, including detailed diffs.
- As tracker system the one offered by SourceForge [reference http://sourceforge.net/tracker/?group_id=102081] is used to report bugs, patches, feature requests.. Notification about tracked issues on the *unicore-notification* mailing-list.
- WIKI [reference http://sourceforge.net/apps/mediawiki/unicore/index.php?title=Main_Page] is used as a workarea: to collect development plans, quick guides for users, admins, developers, FAQs.
- Mailing-lists - used for developers discussions (*unicore-devel*) and release announcements (*unicore-announce*).
- Build and test systems: Atlassian Bamboo is used to manage the build & test process.

3.4.3 Release Categories

Regular releases of UNICORE core components (core servers, workflow, ucc, Eclipse client) are performed at a schedule of approximately three months. The release process combines a t*ime-based release* method with *continuous release* one, trying to release approximately every three months. If

there are no changes, like bugs solved or improvements there's no release. The benefit of having a new release has to outweigh the work associated with it.

To ADD

3.4.4 Workflow:

- a new version of a component is built and tested thoroughly by the developers and any additional "in-house" testers
- the "release candidate" of the component is made available for download on public repository (not SourceForge). It is announced on development mailing list (unicore-devel) and to other stakeholders, e.g. projects.
- there is a 10 working days “voting period“
- in case of bug reports, these are fixed, and the release candidate is updated.
- if no objections are raised and no (further) critical bugs have been reported, the software is released on SourceForge repository.

4. EMI RELEASE MANAGEMENT

4.1. ACTORS AND ROLES

Developer - is responsible for the maintenance of a particular part of the codebase, which includes fixing bugs and adding new features to software components. They can belong to one or more Product Teams, depending on the components they are developing

Product Team (PT) – small teams of software developers, fully responsible for the successful release of a particular software product (or group of tightly related products) compliant with agreed sets of technical specification and acceptance criteria. The individual Product Teams are requested to execute the necessary processes and procedures and respect established requirements and priorities as globally defined and endorsed by the EMI project executive and technical leadership.

Product Team Leader (PTL) - responsible for managing a specific PT, for the design, development, support and maintenance of specific products

Software Quality Control Leader (SQCL) - responsible for the Software Quality Control layer, which ensures both SQA and SQAP are being followed by the development teams. All the releases of EMI components need to satisfy well-defined certification and validation criteria before being included in a stable EMI distribution, sufficient to guarantee to a high degree of confidence that all EMI products meet or exceed the requirements set by EGI and that no regression are introduced.

Technical Area Coordinator: the members of the Project Technical Board responsible for the overall coordination of each of the four technical areas: compute, data, infrastructure and security areas. The TACs are responsible to define together with the relevant Product Team Leaders the technical strategy of their area. They are also responsible to define together with the TD and the other TACs the overall project technical plans. The TAC are proposed by the PD and the TD and approved by the CB.

Release Manager: - responsible for overall coordination. Main duties include: assure the good functioning of the release process, manage release delivery and scheduling, chair the EMT and ensure that its agenda reflects all relevant issues of the day.

4.2. DECISION-MAKING BODIES

Engineering Management Team (EMT) - lead by the Release Manager and composed of the PTLs (or duly appointed representatives), a QA representative, a Security representative, representatives of the Operations teams of the major infrastructures (EGI, PRACE, etc.) and invited experts as necessary. It manages the release process and the "standard' changes, that is all the changes that are pre-approved based on established policies and do not need to be individually approved by the PEB or the PTB, like: software defects triaging and prioritization, technical management of releases, integration issues, and user support request analysis.

Project Technical Board (PTB) - led by the TD and composed of the Technical Area Leaders and a representative and is responsible to assist the TD in defining the technical vision of the project and deciding on specific technical issues. The PTB members are the coordinators of the project Technological Areas.

Project Executive Board (PEB) - responsible to assist the Project Director in the execution of the project plans and in monitoring the milestones, achievements, risks and conflicts within the project. It is led by the PD and is omposed of the Work Package Leaders, the Technical Director and the Deputy Technical Director.

4.3. RELEASE CATEGORIES

The EMI distribution will be organized in periodic major releases, tentatively delivered once a year, providing a good balance between the conflicting requirements of stability and innovation.

An EMI major release is characterized by well-defined interfaces, behavior and dependencies for all included components, available on a predefined set of platforms. What is included in a new EMI major release is defined by the PTB and the implementation of the plan is coordinated by JRA1.

Backward-incompatible changes to the interface or to the behavior of a component that is part of the EMI distribution can be introduced only in a new EMI major release. Changes to interfaces that are visible outside the node where the component runs (e.g. a WSDL) need to be preserved even across major releases, according to end-of-life policies to be defined on a case-by-case basis.

The availability of a new major release of EMI does not automatically obsolete the previous ones and multiple major releases may be supported at the same time according to their negotiated end-of-life policies.

4.3.1 Components Releases

An EMI distribution includes all the components that are developed within the project and that have reached production quality. Within an EMI major release, only one version of a given component is maintained. Four types of releases have been identified for a given component:

- **Major Release**: A major release for a component is characterized by a well-defined interface and behavior, potentially incompatible with the interface or behavior of a previous release. New major releases of a component can be introduced only in a new major release of EMI. The contents of a new major release are endorsed by the PTB and included in the project technical plan. The implementation is coordinated by JRA1.
- **Minor Release**: A minor release of a component includes significant interface or behavior changes that are backwards-compatible with those of the corresponding major release, but it can include new functionality or roll-ups of sets of software fixes not previously released as dedicated revisions. New minor releases of a component can be introduced in an existing major release of EMI. The contents of a new minor release are endorsed by the PTB and included in the project technical plan. The implementation is coordinated by JRA1. If the release is going to be introduced in an existing major release of EMI, the implementation is also supervised by SA1 in order to guarantee that the production quality and the backwards-compatibility are preserved. It is expected that minor releases will be made available a few times per year.
- **Revision Releases**: A revision release of a component includes changes fixing specific defects found in production and represents the typical kind of release of a component during the lifetime of an EMI major release. It cannot include new functionality or change the nominal expected behaviour of the components. Revision releases will be made available very frequently and user will be able to decide whether to apply the revision or not based on the existing impact of the defects on their activities.
- **Emergency Releases**: An emergency release of a component includes changes fixing only high-impact, high-severity bugs like security issues and can follow special shortened release procedures. Emergency releases are made available as needed.

The type of release is reflected in the version of the corresponding package(s), as described bellow.

4.3.2 Versioning Schema

One of the EMI objectives, stated in the DoW, is the inclusion of EMI software products into standard repositories and open source Linux distributions (like Fedora, Ubuntu, etc), with the estimated target of 80% of the client components, selected services based on requirements.

In order to achieve this objective one important activity is to follow some of the main Linux distributions Packaging Guidelines like “Fedora Packaging Guidelines” [R#: http://fedoraproject.org/wiki/PackagingGuidelines) and “Debian Policy Manual” [R# : http://www.debian.org/doc/debian-policy/).

The EMI Packaging and Releasing Guidelines [R#:link to https://twiki.cern.ch/twiki/bin/view/EMI/EmiSa2PackagingReleasingGuidelines] prepared by SA2 in collaboration with SA1 should be used b the PTs when preparing their components.

Some of the guidelines are summaries bellow:

Fedora/RHEL:

Package Naming

- Dash '-' must be used as delimiter of name parts, no underscore '_' nor plus '+' are allowed
- Even if they differ by case, package names conflicting with already existing ones are not allowed
- The SPEC file name should be %{name}.spec
- Subpackages such as doc, devel, etc. should be named with the main name and must use dash '-' as a delimiter. Examples are ssl-devel, httpd-doc.
- Modules:
  - PERL modules should have a name with 'perl-' prepended
  - Python2 modules should have a name with 'python-' prepended
  - Python3 modules should have 'python3' instead of 'python'

Package Versioning

- RPMs versions are: VERSION and RELEASE
- VERSION and RELEASE are delimited by dash '-'
- No dash is allowed in the VERSION
- To use a single SPEC file for different distributions, it is recommended to append the %{?dist} tag in the RELEASE field
- To release bugfixes for old branches, the recommended way is to append a version after the %{?dist} tag in the RELEASE:
  - Release: 1%{?dist} becomes 1%{?dist}.1 so that foo-1.0-1.fc4 < foo-1.0-1.fc4.1 < foo-1.0-1.fc5

EMI will adopt, for components releases, as VERSION-RELEASE numbering convention the typical major.minor.revision(-[age/release]) schema, where:

- increment in major# - reflects a change in the component interface or behavior, that can be backward incompatible
- increment in minor# - reflects a change in the component interface or behavior backward compatible

- increment in revision# - reflects bugs fixes, with no new features
- increment in release# - reflects a rebuild due to change in the external dependencies; changes in the ABI that imply the rebuild of all the clients; an emergency release for component-x.y.z, when the version component-x.y.z+1 is already ready.

4.4. RELEASE CRITERIA

The section describes which will be the criteria that must be met before deployment in the production environment, based on quality assurance recommendations, quality control certification & validation criteria.

According to ITIL, the objective of the Release Management is to ensure that:

- there are clear and comprehensive release and deployment plans that enable the customers to align their activities with these plans
- a release package can be built, installed, tested and deployed efficiently to a deployment group or target environment successfully and on schedule
- a new or changed service and its enabling systems, technology and organization are capable of delivering the agreed service requirements, i.e. utilities, warranties and service levels
- there is minimal unpredicted impact on the production services, operations and support organization
- customers, users are satisfied with the service transition practices and outputs, e.g. user documentation and training.

In order to be sure that these objectives are met, the release process is based on SMART release criteria, where SMART stands for:

- Specific – the criteria is for the product (EMI) at this point in its life-cycle
- Measurable – one can evaluate the software against it
- Attainable – each criterion is achievable
- Relevant - criteria are relevant by evaluating this product against what the customer wants and what developers wants
- Trackable – one can evaluate the state of the criteria during the project

Purpose of having release criteria:

- Clearly specify the criteria that *must* be met for each of the public releases
- Document all the requirements of the customers/users
- Establish when a release is "done"

Some of the general criteria are:

- all code must compile and build for all mandatory platforms
- zero high priority bugs
- for all open bugs, documentation in release notes with workarounds
- all planned QA tests run, at least ninety percent pass

- functionality decided in planning phase is successfully tested on the release candidate
- ready to release by April 30!

to be completed with the specific criteria for each major EMI release [R# reference to SA1 twiki containing the release criteria]

The way to release a component within an EMI release is by mean of a Release Candidate (RC), that will be characterized by the following information:

- List of packages affected by the change and the URL from where they can be downloaded.
- Associated EMI major release.
- Release Notes: non-technical text written in good english giving an overview of the change introduced by the packages.
- List of bugs/feature requests fixed in the packages and the links to the corresponding item in the tracking tool.
- Link to the test report where the test results of the executed tests are included. This should prove that all mandatory tests relevant to the released packages have been executed successfully.
- Link to the relevant documentation (user guides, troubleshooting guides, etc...)
- List of known issues associated with the release.
- Changelog of each of the released packages containing the developer's comments associated with each change.

4.5. RELEASE PROCESS

4.5.1 Coordination Meetings, Comunication Channels

As mentioned before, the EMT is the place where takes place the technical management of releases, discussions on integration issues, and user support request analysis.

Weekly meetings, telephone conferences, are organized on Monday, from 15:00 to 16:30 to discuss:

- status of releases:
  - supported platforms;
  - external dependencies;
  - ETICS-configurations,
  - test & certification;
- documentation:
  - updates in Guidelines for Configuration & Integration, Packaging & Releasing, Change & Release Management, Testing & Certification, Metrics Generation
- review of the RfC (Request for Change):
  - checks whether the information in the RFC is illogical, unfeasible, unnecessary or incomplete.

Meetings are held using the EVO system [Ref.] and the agendas are available under Indico [Ref.]. The mailing-list emt@eu-emi.eu is used for the EMT discussions, communications of meetings. All details are included on the EMT twiki page at [REF]

4.5.2 Tracking-Tool (Jira), Issues Fields

4.5.3 External Dependencies

One of the issues of having to integrate 4 middleware stacks is that the external dependencies used by different middleware distributions are of different versions and are taken from different sources The solution found by the Configuration & Integration Task-Force is that all components will try to use default versions from OS YUM/APT repositories and EPEL.

A common list of dependencies for all components was created [Ref] and is used to install them on the VM images used by the ETICS-build system.

To add external dependencies, as this should be added also in the ETICS project configuration, PTs should follow the steps bellow:

- A PT requests to have an external dependency with a certain version in the ETICS project configuration.
- The PT confirms that the dependency + version can be installed from OS or EPEL.
- If the dependency comes from another repository or the requested version is not available:
    1. The PT justifies the need for it in the EMT
    2. The EMT approves the distribution of the external dependency from the emi-third-party repository.
    3. The PT is then responsible for maintaining the external dependency in ETICS.
    4. The information on the special external dependencies is added on the Release Candidate task, in JIRA
    5. The affected components are released to production with the external dependency.
    6. If a package is not available in the right package format (RPM or DEB) and it is required as dependency, the requester PT will be responsible of creating the proper package, building the package from source and making it available as an ETICS external.

4.5.4 Build Process

Aiming at achieving the objectives present in the DoW:

- a target of 80% of clients in mainstream Linux distributions
- Reduction of redundant components
- Software development tools aligned across activities, partners and middleware stacks

- After a transition period the entire project must be aligned on the same procedures and tools.

the Integration & Configuration Task-Force arrived at the following conclusions and prepared the Configuration & Integration Guidelines [Ref]:

- A single tool will be used to perform integration builds from source (nightly builds). All build tools will be supported and integrated via this unified integration system.
- Even though ETICS cannot be considered the optimal tool for each environment, it is the only one able to handle an integration build from source of all the four middleware distributions. Therefore it will be used to produce a first unified integration system.
- EMI is to use a common build, test and QA system as much as practically possible. Currently the system of choice is ETICS. All components to be released as part of EMI must have configurations in ETICS
- EMI is released on two sets of platforms, mandatory and optional. Mandatory platforms must be supported by all components in EMI, optional platforms can be supported by individual PTs depending on needs of technical capabilities. Currently EMI 0 and 1 are going to be supported on RHEL5 and compatible platforms and Debian 5 and compatible platforms on 32 and 64bit platform
- All components in EMI must be provided in the standard packaging formats typical of the supported mandatory platforms. Currently this means rpms for RHEL5 and compatible platforms and debs for Debian 5 and compatible platforms
- The EMI Repository must contain always non conflicting packages and only packages that are not commonly available on standard OS repositories or other mainstream repositories such as Fedora or EPEL

4.5.5 Delivery Strategy(EA announcements, EMI Repository upload)

The section will describe the release process, from the coordination meetings (EMT) where decisions on features list, priorities, will be taken, to the tracking-system (issues fields description), description of the build process, management of external dependencies, and the delivery strategy – repository update, communication with EA-sites, EGI and other DCIs.

4.6. RELEASE SCHEDULE

4.6.1 Time-Based Release Schedule, Maintenance Schedule

4.6.2 3-years Release Schedule

4.6.2.1 Feature List

4.6.3 EMI First Release Schedule

4.6.3.1 Feature List

4.6.3.2 Release Schedule

The section will contain the motivation of time-based releases, the maintenance (EOL) schedule, a list of the features that we'll try to achieve during the three major releases and a detailed feature list and schedule for the first-year EMI-release.

5. INTERNAL RELEASE EMI-0

EMI-0 is an 'exercise' release designed to understand how to apply the agreed procedures, find any problem about tools and processes and in general fine tune the EMI software engineering process before the EMI 1 release. The outcome of EMI 0 is not expected to be released to external users. The goal of EMI 0 is to prepare a consistent, coherent repository of non-conflicting packages by the end of October 2010 without any specific commitment on functionality.

6. CONCLUSIONS

<The conclusion is not a repetition or summary of the content. It should not be again an executive summary. The conclusion is a brief description of the outcome, consequences or further work to be done beyond the work described in the document>.