Advancements in EMI Testing Infrastructure Implementation

Danilo Dongiovanni[1]
INFN-CNAF
Viale Berti Pichat 6/2 40100 - Bologna, Italy
E-mail: danilo.dongiovanni@cnaf.infn.it

Tomasz Wolak
CERN
Geneve, Switzerland
E-mail: twolak@cern.ch

Bjorn Hagemeier
Forshungszentrum Julich GmbH
52425 Juelich, Germany
E-mail: b.hagemeier@fz-juelich.de

Marek Kokan
Faculty of Sciences, UPJS
Jasenna 5 040 01 KOSICE - Slovakia
E-mail: marek.kocan@upjs.sk

Christian Bernardt
DESY
Notkestraße 85 D-22607 Hamburg, Germany
E-mail: christian.bernardt@desy.de

Frantisek Dvorak
CESNET
University of West Bohemia, Univerzitni 20, 306 14 Plzen, Czech Republic
E-mail: valtri@civ.zcu.cz

[1] Speaker

ABSTRACT

The European Middleware Initiative (EMI) Project has succeeded in merging into a single release (EMI 1 Kebnekaise) more than fifty software products from four major European technology providers (ARC, gLite, UNICORE and dCache). To satisfy end user expectation in terms of functionality and performance, release process implements several steps of certification and verification. The final phases of certification are aimed at harmonizing the strongly inter-dependent products coming from various development teams through parallel certification paths. The role of the EMI Testing Infrastructure is to provide operational and infrastructural resources to implement inter-component certification phases and involve EMI end users in early testing or preview activity. Moving from a brief introduction to challenges faced during the first EMI project year, the work presents operational and infrastructural solutions put in place to setup the EMI Testing Infrastructure and maximize EMI product exposure to different testing scenarios and EMI products user communities.

EGI Community Forum 2012 / EMI Second Technical Conference,
Munich, Germany
26-30 March, 2012

1. Introduction

The European Middleware Initiative (EMI) [1] project merges into a single release more than fifty software products from four major European technology providers (ARC [2], gLite [3], UNICORE [4] and dCache [5]). The release process is designed to allow independent and parallel work across more than thirty product teams. Periodically, software products resulting from this parallel work become part of a single harmonized EMI release update. Therefore a centralized phase of integration certification must take place. The place where this harmonization and certification happens is the EMI inter-component testing infrastructure. Section 2 will frame this activity within EMIs quality assurance work package.

Inter-component testing infrastructure is meant to be the first central point of contact among different EMI products, i.e. the place where each products functionalities and expected behaviour are tested against other related EMI products. This is implemented by permanently deploying instances of both production and release candidate versions of all product components for every EMI release. Hence an evolving snapshot of all released and upcoming versions of all EMI products is provided to product team developers. Configurable central information system instances, publishing resources in the testbed, assure flexible and dynamic creation of testbed subset views. Section 3 presents recent evolution in the implementation of this infrastructure and relative operational facilities/procedures, with a focus on integration testing approach implemented and automation testing. Additionally Section 4 will describe the collection of activities put in place to let EMI partners and user communities preview EMI products. In particular, the evolution of a large scale acceptance testing infrastructure model in order to provide user communities and developers with more flexible opportunities to setup specific scenarios. Also, the chance for user communities to test EMI products were increased through preview campaigns, involving user community contribution in testing effort rather than just hardware contribution, and hosting demo and training events on the testing infrastructure. These activities implied the implementation of flexible operational solutions for infrastructure provisioning.

Section 5 summarizes the work providing some remarks and lessons learnt during the first two years of the EMI project activity and outlines planned future developments.

2. Role of central testing facilities as part of EMI release cycle and quality assurance activities

The evolution of EMI software products in order to fix software errors or implement new features follows a defined release cycle, resulting in both monthly release updates (minor releases whilst not breaking backward compatibility) and yearly major releases. To these periodical releases we add revision (fixing defeats without introducing new features) and emergency updates (fixing problems with top priority, generally related to security).

The release process periodically cycles over five macro phases:

- *phase 1*-) Requirements analysis phase: inputs collected from EMI user communities representatives (EGI [6], WLCG [7]) are translated into accepted technical requirements;

- *phase 2*-) Development and test planning phase: technical requirements from phase 1 bring to new development and test plans;
- *phase 3*-) Development, testing and certification phase: new products versions are developed according to test plans and tested by product teams;
- *phase 4*-) Release certification and validation phase of new products candidates;
- *phase 5*-) Release and maintenance;

All EMI software products undergo this release cycle many times per year. Minor release cycles are asynchronous for different products, meaning that each product is independent from other products in its development and certification provided that it does not break backward compatibility. The project structure reflects this approach with more than thirty product teams working in parallel according to agreed policies.

The EMI quality assurance work package is in charge of those activities aimed at harmonizing the parallel work of the developer product teams to obtain as output a single homogeneous EMI release. Therefore, among quality assurance duties we have the definition and monitoring policies, definition and collection of metrics and keys performance indicators (KPIs), quality control verification and reporting, the provision of common tools for products building and the implementation of common and shared infrastructural and operational resources for product inter-component and large scale testing. The present work focuses on this last working area in EMI quality assurance, which is strictly related to the phase 4 and 5 of release cycle. Given the framework described above, we can summarize the role of central testing infrastructure team as a provider of all facilities and certification activities assuring that the sum of components certified in isolation constitutes an EMI release of products deployable from single repository and consistently interoperating.

2.1 Overview of EMI Products certification testing

Before presenting the EMI central testing facilities and operations in details, it is worth to clarify how each EMI software component life cycle maps into EMI release life cycle. Each EMI component follows an independent path from other components life cycle during the definition of requirements, development/test planning, source coding, build until the component certification in isolation. Then, after certification in isolation has been accomplished, the various component release paths must intersect in order to verify that all components can consistently interoperate. This means that the logically unitary phase of component certification, aimed at verifying the expected functioning under production environment, is actually split in two separate steps across EMI release cycle phases 3 and 4. Moreover, component certification in isolation during phase 3 is performed on product team resources while inter-component testing performed during release phase 4 occurs on central testbed resources.

To give an overview of the types of tests performed in each phase we mention:

- Release phase 3: static code analysis, installation tests during repackaging phase in the mock image to verify run time dependencies, deployment tests on product teams local resources, unit tests, functionality and regression tests of the component in isolation;
- Release phase 4: deployment tests on central resources, functionality tests validating EMI components mutual interaction;

- Performance and scalability tests are not mandatory and may occur both in phase 3 and 4 of EMI release cycle. Scalability tests are partly implemented through activities described in Section 4.

The results of all tests are finally gathered into a unique test report.

Concerning the way these tests are performed, it should be noted that the level of automation in testing tend to quickly decrease when passing from static code analysis to scalability tests and it is not homogeneous across EMI products. The subject is treated in Section 3.3 in full details, but we can anticipate that only static code analysis and mock installation tests are fully automated as a post-build step in the EMI central build system tool [30]. Unit and functional tests are mostly automated but implemented with different technologies depending on the product considered. Deployment tests on the central testbed are manually performed as well as inter-component tests, except those implemented by SAM-Nagios [11] probes. Performance and scalability tests are mostly not automated.

3. EMI Infrastructure and facilities for internal release testing

Two main categories of tests derive from the goal of ensuring that all EMI components are deployable from single repository and consistently interoperating: release deployment tests and inter-component tests.

3.1 Release deployment testing

After a quality check verification step, formally controlling product compliancy with agreed release policies and guidelines, the release components candidate for the considered update are deployed on the inter-component testing infrastructure. This deployment test is independent from the one performed by a product team during certification and provides the Release Manager with information on the actual status of a release, which is particularly needed when preparing a major release and having no stable components.

Most common product deployment scenarios are adopted to reproduce production environment conditions and also updates from previous production versions are tested. These deployment tests require the cross-configuration of product instances geographically distributed across seven partner sites with different farming solutions. For this reason, a common centralized system for automated deployment is not implementable. Nonetheless some solutions for automated deployment based on bash scripting or puppet farming administration tool [49] are under development.

Quality verification and testbed deployment results are mandatory ingredients for the release manager decision on accepting each candidate product in the next EMI update. Fig.1 illustrates details of these specific release cycle steps.

3.2 Inter-component testing infrastructure

As shown in Fig.1, all component release candidates must pass an inter-component testing certification step to enter the next EMI update. This inter-component testing is performed on EMI central inter-component testing infrastructure instances.

In this context, we recall the definition of inter-component testing as the part of certification of an EMI software product where the product functionalities and expected behaviour is tested against other EMI software products interacting with the product considered. Integration testing takes place after the certification testing of the product in isolation has been successfully carried out and reported in a public test report.

To understand the solution implemented for the central testing infrastructure we enumerate the required categories of inter-component tests:

1. Integration testing within a minor release (i.e. no backward compatibility broken), so that a Release Candidate (RC) component can be tested against other services in production. This test implies a distributed testbed of production services available for each middleware stack (Arc, dCache, gLite, Unicore), with possibly multiple instances for central services. This could also (rarely) imply cases of RC versus other RC or RC versus (RC + production).
2. Integration testing for a major release (where it is allowed to have new features or backward compatibility broken for many services). This implies a testbed of RC components versions deployed for each middleware stack.
3. Integration testing among middleware stacks (ARC/dCache/gLite/Unicore). This testing scenario is normally covered by testbeds defined in points 1 and 2 above, but could also imply specific testbed setup for experimental service versions.

The implemented infrastructural solution for the testing scenarios identified above is represented in Fig.2.

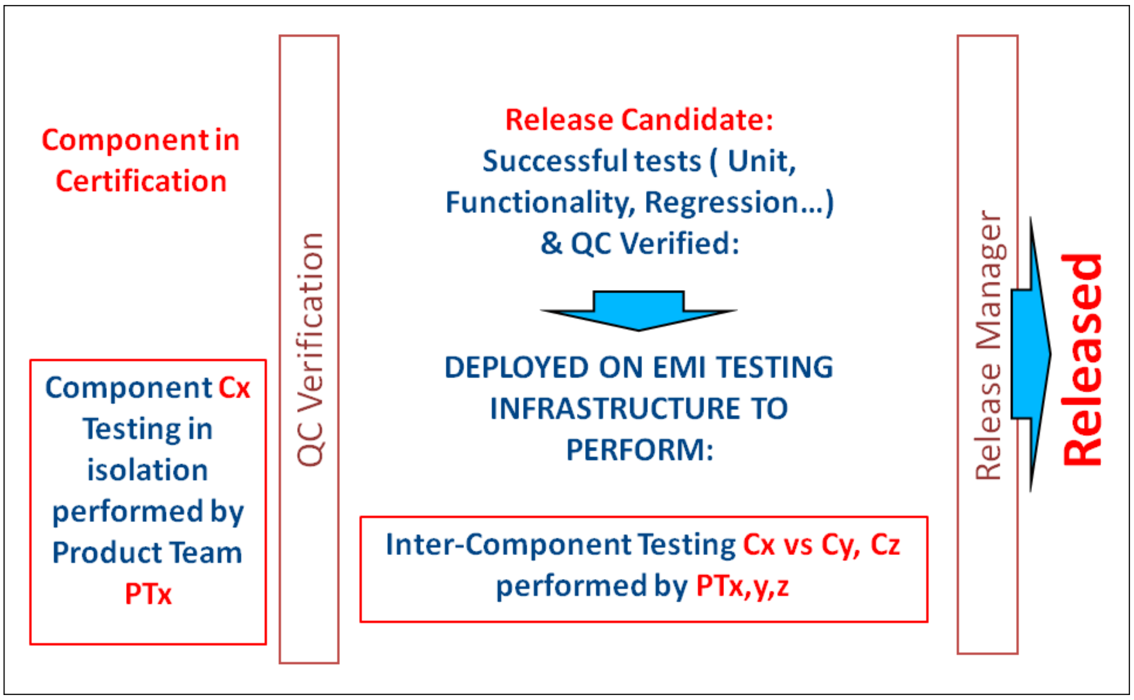
Component in Certification
Component Cx Testing in isolation performed by Product Team PTx
QC Verification
Release Candidate: Successful tests (Unit, Functionality, Regression…) & QC Verified:
DEPLOYED ON EMI TESTING INFRASTRUCTURE TO PERFORM:
Inter-Component Testing Cx vs Cy, Cz performed by PTx,y,z
Release Manager
Released

Figure 1: EMI release components successfully passing the phase of certification in isolation are verified for congruency to process policy and deployed on the inter-component testing infrastructure. On the resulting testbed integration testing certification can be performed by all product teams concurrently. Finally, all component passing release quality criteria enter new release update.

The current implementation of inter-component infrastructure counts for more than 180 instances providing a snapshot of: i) pre-EMI products from the four partner middleware converging into EMI (to test backward compatibility with user interface, batch systems and worker nodes still existing in communities sites); ii) EMI-1 Keiknekaise [8] production version release; iii) EMI-2 Component Release candidate versions other than some instances for tool testing or platform testing.

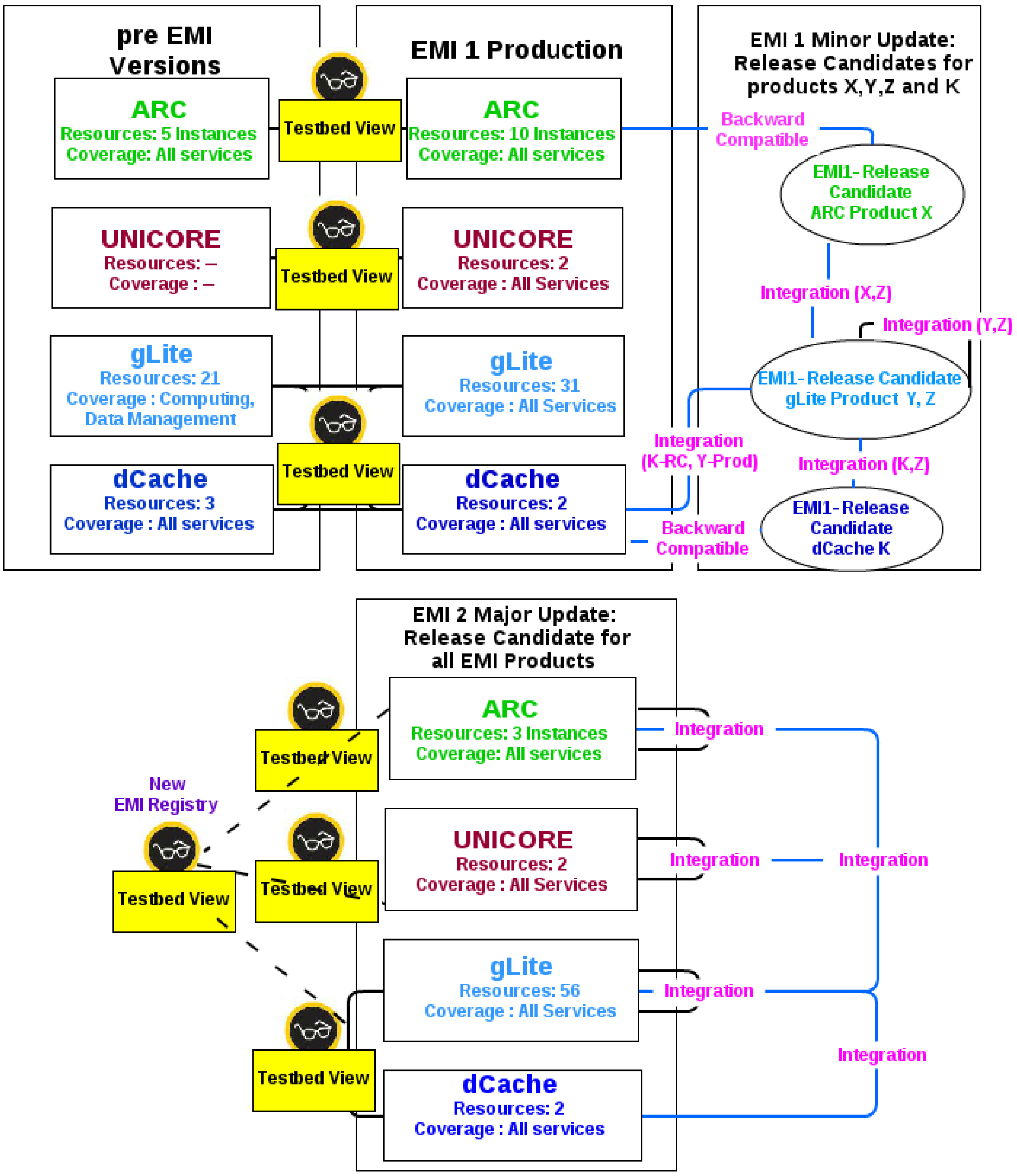
pre EMI Versions
ARC Resources: 5 Instances Coverage: All services
UNICORE Resources: – Coverage : –
gLite Resources: 21 Coverage : Computing, Data Management
dCache Resources: 3 Coverage : All services
Testbed View
EMI 1 Production
ARC Resources: 10 Instances Coverage: All services
UNICORE Resources: 2 Coverage : All Services
gLite Resources: 31 Coverage : All Services
dCache Resources: 2 Coverage : All services
EMI 1 Minor Update: Release Candidates for products X,Y,Z and K
Backward Compatible
EMI1- Release Candidate ARC Product X
Integration (X,Z)
Integration (Y,Z)
EMI1- Release Candidate gLite Product Y, Z
Integration (K-RC, Y-Prod)
Integration (K,Z)
EMI1- Release Candidate dCache K
Backward Compatible
EMI 2 Major Update: Release Candidate for all EMI Products
ARC Resources: 3 Instances Coverage: All services
UNICORE Resources: 2 Coverage : All Services
gLite Resources: 56 Coverage : All Services
dCache Resources: 2 Coverage : All services
Integration
New EMI Registry
Testbed View

Figure 2: Inter-component testing infrastructure. Release Candidates components are tested for integration both with production and release candidate versions of other integrated EMI components. Additional information services or registry services instances aggregate testbed instances into testbed views which can be modularly combined.

These instances are geographically distributed across 7 EMI participant institutes (CERN, CESNET, INFN-CNAF, DESY, JUELICH, KOSICE, NIIF). A flexible grouping strategy of product instances in modular *testbed views* is implemented by information system services for production and RC versions for different middleware[2]. Building custom testbed views including both EMI central resources and product team local resources is also possible by composing testbed views as for example by republishing resources in cascade across multiple information system services. This approach is used also in preview activities where we generally create ad hoc testbed views merging EMI testing infrastructure resources with production resources.

As operational facilities we also provide two Virtual Organizations (testers.eu-emi.eu and testers2.eu-emi.eu) and a support unit through GGUS [9] support portal. Detailed documentation on both infrastructural and operational resources can be found at [10].

3.2.1 Advancements on infrastructure implementation and activities

The first relevant advancement since EMI-1 is the increased dimension of testbed which doubled between EMI-1 and EMI-2. This was mainly due to both the cumulative effect of assuring a coverage for all supported and candidate EMI releases and to the increased number of platforms supported from EMI-2 on. In fact, while EMI-1 release was delivered on Scientific Linux 5 x86/64 platform only, EMI-2 will support Scientific Linux 5 x86/64, Scientific Linux 6 x86/64 and Debian 6.01 platforms. To reduce the impact of multiplatform deployment on the testbed dimension, whenever possible we opted for a serialization of components testing on each platform. The release cycle was accordingly extended to have a three weeks (1 platform per week) deployment and inter-component testing cycle. This allows for virtual machine disks replacements each week without changing services endpoints and cross configuration. Cross platform testing is performed only when expressly required by product teams.

During EMI project second year also the scope of activities related to release testing has significantly increased. In fact, in addition to deployment/inter-component tests for the 17 EMI-1 Updates delivered and the EMI2 Release Candidate testing, the testbed infrastructure staff was involved in bridge activities between product teams and release management or between EMI developers units and EMI external partners or user community representatives. Among these activities it is worth to mention: i) feedback provisioning on product documentation; ii) support to WLCG user community to plan migration to EMI middleware from pre EMI middleware stack by testing and documenting this migration; iii) release level bug fix and or workaround verification; iv) planning of test modalities for SAM-Nagios [11] product probes (released by EMI) on new SAM-NAGIOS service framework (released by EGI).

3.2.2 Advancements on inter-component testing approach

One of the main challenges faced during EMI-1 major release certification was the definition and implementation of an efficient inter-component integration testing approach.

The following main criticalities were identified:

1. Time Coordination in component release certification. According to EMI release model (see Section 2) software components follow independent and

[2] Note that a single registry or information system service for all EMI components is currently under development.

asynchronous update paths (except for major releases). For mutually integrated products (ex. X,Y) this implies either component X to continuously check the integration with product Y at each new build or product teams Px and Py to coordinate the delivery and certification in the same time window.

2. Definition of Integration test perspective and accountable product team. In fact very different integration tests can be conceived between two given products *X* and *Y* depending on how representative is the considered "testing perspective”. As an example a team producing CREAM compute element could test integration with ARGUS authorization just with a direct submission. But the end user testing perspective would be testing that the whole job management works when having a site with WMS, ARGUS, CREAM, glexec-WN, mpi, DPM and STORM services consistently cross-configured. The ultimate goal is clearly to have a reasonable coverage of production environment perspectives. But a side effect of widening the client perspective for the integration test is the proportionally increasing difficulty in defining a policy to determine which product team should be then considered accountable (in charge of performing the test and filling the test report with results) for that integration test.

A continuous integration approach on a separated testbed deploying only release candidate versions of products had been initially identified as a project objective to solve the time coordination criticality. In our context *continuous integration* would mean: i) to deploy on the central testbed any successfully built new version of all EMI component and ii) to test them for integration. But this approach was finally deemed to be not implementable in practise without automating the full build-deployment-testing process. The reasons preventing for process full automation resides in the actual poor level of EMI cross product integration and harmonization especially when concerning technologies chosen for testing automation (see section 3.3). Moreover it should be noted that cross product integration tests are generally difficult to automate as an extension of a single component/products test-suite, mainly because a distributed environment setup involving several services is required by definition. On the other hand the convergence of all products on a single testing technology was not a development priority for EMI project.

Therefore the solution adopted for EMI 2 major release was to implement an inter-component testing approach called Integration Tests campaigns. By Integration test campaigns we mean the definition of a specific set of integration tests to be run between the release candidate version of the components deployed on the central testing infrastructure within a predefined time window in the release cycle. This approach assures time coordination in the inter-component testing activities across product teams. An Integration Testing Task Force was created with the purpose of defining a list of integration tests covering most common end-user perspective use cases. Inter-component testing campaigns follow the deployment of release components for each platforms in a 3 week cycle with 1 week per supported platform.

Table.1 summarizes the inter-component integration tests already implemented. All planned test are reported at [12]. Other than the products/software-area involved in each integration test, Table.1 also reports whether the test is automated or not. Only data management integration tests have been implemented exploiting the specific test-suite of one of the component to test (ex. T-StORM [31] for testing data management components interoperation). . More details about automated testing and monitoring solutions are provided in next section.

Table 1: Integration tests in place and products involved in tests	Authentication/ Authorization	Job Management	Data Management	Infrastructure	Automated
(CEMON,Argus),	X				-
(Argus,VOMS, CREAM); (glExec-WN,CREAM)	X	X			X
(EMIR, UNICORE/X, A-REX);		X		X	-
(gLite MPI,WMS,CREAM,WN); (gLiteWMS, gLiteLB)		X			X
(A-REX,CREAM, WMS, BDII); (ARC compute CLI,UNICORE/X,CREAM CE,A-REX)		X		X	-
(dCache,DPM,StoRM,FTS)			X		X
(A-REX,DPM,dCache,StoRM,LFC)		X	X		-
(StoRM,DPM,dCache,ARC gridftp server)			X		-
(lcg_util,DPM,dCache,StoRM)			X	X	-
(ARC data clients, LFC, dCache, DPM, StoRM, VOMS)	X		X		-
(Apel,CREAM);		X		X	-
(VOMS,BDII);	X			X	-
(ARIS, Top BDII)				X	-

3.3 Automation: test-suites and monitoring tools

In this section, we describe the monitoring solutions adopted for the testing infrastructure and the perspective to increase the level of automated testing coverage of product deployed in the testbed.

3.3.1 Automated testing approach survey for EMI products

As mentioned in section 3.2.1, the main obstacle to efficiently implement the continuous integration approach in EMI products inter-component testing certification is the lack of a common framework for automated testing.

Table 2 reports the results of an internal survey on automated testing implementation status. The approach to testing automation varies across EMI products, with some sub-group homogeneity due to legacy reasons. By far, gLite represents the most fragmented reality across products (Table.2 reports a not exhaustive sample for EMI gLite products, details available at [36]). The reason for this persistent heterogeneity resides in the fact that EMI project focus in first two years was on forcing products to have a common build system and adding some key functionalities/products to increase integration across middlewares (EMI Execution Service or EMI Registry service are an example).

It should be noted that the ETICS build and integration tool [30] exploited for EMI middleware build, implements static analysis of the code or packages. For example, the following plugins do static checks as counting number of lines of code for each language, verifying compliance of the packages with Fedora guidelines and checks compliance with IPV6 or presence of tests, results and coverage: SLOCCount, Rpmlint, IPV6, PyUnit, PyCoverage, PMD [41], JCCN, FindBugs [42], CheckStyle [43], CCCC [44], Pylint [45], CCppCheck [46]. Also ETICS checks build time / run time dependencies Mock [47] or Pbuilder, so that part of deployment testing is automated at build time with ETICS for the whole EMI project.

3.3.2 SAM-Nagios Monitoring

Concerning EMI Testbed monitoring we can distinguish 2 major levels: i) *low-level monitoring* (hardware, networking, operating system); ii) *service-level monitoring* (checking availability and validating functionality of Grid middleware services).

The minimum for *low-level monitoring* is checking network availability of machines hosting a service, often realised as generic ping (ICMP) probe checking if the host is responding. Low-level monitoring solution is NAGIOS [13]. *Service-level monitoring*, much more important for a Grid Testbed, needs a more complex solution. The reason is that a probe checking a given Grid service requires relatively complex environment for execution, which normally can be found e.g. on a User Interface node. This includes environment settings allowing to find service endpoints, clients and/or libraries providing access to Grid services (command-line clients or an API) and also Grid user credentials (Grid proxy) for accessing services which require GSI authentication. In the times of pre-EMI (EGEE project [14]) the monitoring solution (at least for gLite middleware) was EGEE-NAGIOS, monitoring framework based on NAGIOS, gLite User Interface and a number of other software components, including YAIM[15] and NCG[16] configuration tools, which allow semi-automatically to have the framework operational.

In the era of EMI and EGI, EGEE-NAGIOS has evolved into SAM-Nagios, with a plan to modify current monitoring framework to cover all services available in the new EMI middleware. At present time, the SAM-Nagios framework is developed by EGI project, while EMI product teams are in charge of producing the probes for the middleware services they develop. Testing integration of service probes and SAM-Nagios framework for EMI is done in the EMI inter-component testing infrastructure. At the time we write, 25 out of 33 EMI services probes are ready and the work on SAM-NAGIOS framework is still ongoing to work with the *emi-release* metapackage. Details about SAM-NAGIOS instances deployed at partner sites can be found on the testbed inventory page [17].

It is interesting to notice here that part of the integrations tests reported in Table.1 are implemented by SAM-Nagios probes, exploiting the command line interface coming with user interface service. As an example we mention the gLite job management SAM-Nagios probe which tests the integration between CREAM, BDII, WMS, VOMS, MyPROXY, LB, DPM by monitoring the full job cycle from selecting resources to submitting a job involving data management operations. On the other hand, the testing infrastructure team is also involved in the validation of SAM-Nagios probes. This means that some SAM-Nagios probes failures does not forcedly imply a failure on monitored services but could reveal problems on the monitoring system itself.

Table.2 : Survey on automated testing across EMI products.	**Static Code Analysis** [1,2]	**Deployment Test** [1,2]	**Unit Test** [2]	**Functionality Tests** [2]	**Integration Tests** [2,3]	**Performance Tests** [2,3]	**Scalability Tests** [2,3,4]	**Exploited Testing Framework**
ARC	(A,T) [18]	(- ,M)	(A,T)	(A,T) [19]	(- ,M)	(- ,M) [20]	(-,-)	Python scripting
dCache	(A,T)	(- ,M)	(A,T)	(A,T)	(- ,M)	(- ,M)	(-,-)	Jenkins [21]
UNICORE	(- ,M)	(- ,M)	(A,T) [22]	(~ ,T)	(- ,M)	(~ ,M)	(-,-)	Atlassian Bamboo [23]
gLite (DPM,FTS, LFC)	(- ,M)	(A,T)	(A,T)	(A,T)	(~ ,M)	(- ,M) [25]	(- ,M)	Saket [24]
gLite Job Computing (CREAM, WMS)	(-,-)	(- ,M)	(~ ,M) [26]	(A,M) [27]	(~ ,M)	(- ,M)	(~ ,M)	Robot Framework [28]
Glite (CaNL,gridsite, LB, proxyrenewal)	(-,-)	(A,T)	(A,T)	(A,T)	(- ,M)	(~ ,M)	(-,-)	Scripts [29], ETICS [30]
StORM	(-,-)	(A,M)	(-,-)	(A,M)	(~ ,M)	(~ ,M)	(-,-)	T-StORM[31]
gLiteMPI	(-,-)	(- ,M)	(A,T)	(~,T)	(- ,M)	(-,-)	(-,-)	Shunit [32]
gLite infosys	(-,-)	(A,M)	(A,M)	(A,M)	(A,M)	(A,M)	(A,M)	Bash Scripts
ARGUS	(-,-)	(- ,M)	(A,T)	(A,M)	(- ,M)	(A,M) [34]	(A,M)	Bash, Python Scripts, Grinder [33]
APEL	(-,-)	(- ,M)	(A,M)	(A,M)	(-,-)	(-,-)	(-,-)	Bash Scripts, JUnit [35]
VOMS	(-,-)	(- ,M)	(A,M)	(A,M)	(-,-)	(-,-)	(-,-)	Dejagnu [48], JUnit [35]
Hydra	(-,-)	(- ,M)	(-,M)	(-,M)	(-,M)	(-,-)	(-,M)	Bash Scripts
Trustmanager	(-,-)	(A,M)	(A,M)	(A,M)	(-,-)	(-,M)	(-,-)	JUnit [35]
AMGA	(-,-)	(- ,M)	(A,T)	(A,T)	(-,M)	(-,M)	(-,-)	Bash Scripts
WNodes	(-,-)	(-,-)	(~,-)	(A,M)	(-,-)	(M,-)	(-,-)	Python Scripts
ETICS – All EMI products	(~,T)	(~,T)[5]						ETICS [30]

*Legend (**A / ~ / -**: Automated / Partially / Not Automated || **T / M**: Automatically / Manually triggered, - Not Performed)[Reference Link]*

Notes:

1. *Tests performed on product teams local resources;*
2. *Tests performed on ETICS build system central resources;*
3. *Tests performed on EMI Inter-component testing resources;*
4. *Tests performed on EMI Partner large scale testbed*
5. *ETICS performs an installation test during repackaging phase in the mock image to verify run time dependencies. This test does not replace the full deployment test.*

Therefore, not to blindly trust SAM-Nagios probes results, a series of functionality tests reproducing common job/data management operations, authorization/authentication tests, information service retrieval and publication checks are also performed at each update testing cycle. Part of these checks have been automated by scripts running on central testbed User Interface.

4. Infrastructure and facilities for EMI partners and user community

Being aware of the fact that the production environment is the ultimate realistic test for product quality assessment, part of our effort is devoted to involving end-user communities in previews of EMI products or specific scenario testing campaign.

Also preview campaigns were proposed to user communities in proximity of EMI major release, with the purpose of receiving highly qualified early feedback on EMI products. On the EMI side we provided deployment expertise and infrastructural facilities such as a common Virtual Organization and Information system services to collect the network of deployed products. EMI1 release preview campaign [37] involved 16 EMI partners providing useful feedback on documentation. The evolution of this activity was twofold: some partners signed a Memorandum of Understanding with EMI projects as part of works with EMI program [38] to have access to EMI testing infrastructure, other partners contributed to the EMI large scale infrastructure for acceptance tests [39].

The goal of large scale infrastructure is to provide a testbed for scalability and interoperability testing of EMI components. At the present time we have 15 instances deployed across four sites. Initially we proposed two alternative approaches: 1.pre-deployment across voluntary sites of product release candidate versions; 2. Demand (from EMI developers to test specific scenario) and Supply (from volunteers partner sites). The first approach resulted in poor participation by user communities also involved in similar staged roll-out activities. So we plan to focus on early testing of specific scenarios but allowing for *demand* also to come from users bringing real use cases experience.

The outreach activity consists in the provisioning of a training and dissemination infrastructure with EMI stable products. We currently have in place a training infrastructure deploying stable versions of EMI products, configured with fake certificates facilities by the GILDA certificate authority. The infrastructure has also been successfully exploited for demo activities from user communities (best demo winning at EGI technical forum 2011, Lyon). As perspective work we are starting a collaboration with Future Grid [40] to host a EMI product snapshot to hold training/demo sessions. The feasibility of an “on demand” instantiation approach for services is under investigation in that context.

5. Conclusion and future plans

The present work reported on the status of the EMI centralized testing infrastructure with related operational resources with a focus on the last advancements achieved in order to better support both EMI product integration testing and release quality verification steps.

At the same time we described the criticalities emerging from the challenging goal of merging in homogeneous release updates the asynchronous and continuous changes occurring across more than 50 EMI products. The solutions implemented resulted as a drawback between the efficiency of release process and the necessity of exploiting most of the existing legacy testing technologies adopted by the various products/middleware merged into EMI. In particular the great heterogeneity of technologies adopted across products (described in Section 2.4) constitutes the main obstacle to achieve full automation in the certification process (deployment, configuration, testing) as simple extension of products certification cycle. This leaded to the decision of replacing the continuous integration testing approach with a inter-component tests campaigns during the EMI projects second year. The release process was also changed accordingly.

We also presented the preview activities promoted by the testing infrastructure team to involve EMI external partners into quality testing of EMI products. In the past year the number of testbed users and contributors increased as well as the differentiation of activities involving external EMI product users community (large scale testbed, EMI preview campaigns, work with the EMI program and training and dissemination). These results witness a better fulfilment of the EMI testing infrastructure ultimate mission, e.g. exposing EMI products to the highest possible number of usage modalities.

EMI year 3 activity will mainly focus on automating the defined set of integration tests and consolidating the automate deployment tools which are currently under development. Also in the next few months new EMI services probes will be integrated in SAM-NAGIOS framework, implying some validation activity to be performed on the EMI testing infrastructure. We also plan to increase the central testbed exposure to all possible existing automated testing facilities (by product teams and/or external user community).

Some new products being released with EMI2 will significantly increase the level of cross-middleware integration. As an example, the release of EMIR, the common registry to all EMI services, will allow to merge the current per middleware (ARC, gLite/dCache, UNICORE) *testbed views* into a unique cross-middleware view. Also the full deployment of a common execution service interface (EMI-ES) will simplify the implementation of automated integration tests for job management. Finally we mention the preview and outreach activity which will naturally derive from the release of these new products so highly impacting production environment.

Acknowledgements

This work is partially funded by the European Commission as part of the EMI Project under the Grant Agreement number INFS0-RI-261611.
We want to thank EMI product teams supplying information about automated testing as well as Duarte Bacelar De Begonha De Meneses and Andres Abad Rodriguez (CERN IT group) for providing information about ETICS tool.

References

[1] European Middleware Initiative http://www.eu-emi.eu/
[2] ARC Project http://www.knowarc.eu/
[3] gLite Project http://www.glite.eu/
[4] UNICORE Community http://www.unicore.eu/
[5] dCache Project http://www.dcache.org/
[6] European Grid Initiative http://www.egi.eu/
[7] Worldwide LHC Computing Grid (WLCG) http://lcg.web.cern.ch/lcg/public/
[8] EMI 1 release Kebnekaise http://www.eu-emi.eu/emi-1-kebnekaise
[9] Global Grid User Support http://www.ggus.org/
[10] EMI Inter-component testing Infrastructure https://twiki.cern.ch/twiki/bin/view/EMI/TestBed
[11] SAM-NAGIOS monitoring https://tomtools.cern.ch/confluence/display/SAM/SAM-Nagios

[12] Inter-component testing task force
https://twiki.cern.ch/twiki/bin/view/EMI/EmiJra1TaskForceIntegrationTesting
[13] Nagios monitoring framework http://nagios.org/
[14] EGEE Project https://twiki.cern.ch/twiki/bin/view/EGEE/WebHome
[15] YAIM configuration tool https://twiki.cern.ch/twiki/bin/view/EGEE/YAIM
[16] EGEE Nagios https://twiki.cern.ch/twiki/bin/view/EGEE/GridMonitoringNcgOverview
[17] EMI Testbed Inventory monitoring solution
https://twiki.cern.ch/twiki/bin/view/EMI/EMITestbedInventory#Testbed_monitoring
[18] ARC Revision Tests http://arc-emi.grid.upjs.sk/revisionTests.php
[19] ARC Functionality Tests http://arc-emi.grid.upjs.sk/functionalTests.php
[20] ARC Performance Tests http://wiki.nordugrid.org/index.php/Performance_testing
[21] Jenking Integration Testing Framework http://jenkins-ci.org/
[22] Unicore Testing Framework https://unicore-dev.zam.kfa-juelich.de/bamboo/telemetry.action
[23] Atlassian Bamboo Continuous Integration Framework
http://www.atlassian.com/software/bamboo/overview
[24] Swiss Army Knife for ETICS Testing (SAKET) http://svn.cern.ch/guest/saket/trunk/
[25] SAKET performance Test Suite https://svnweb.cern.ch/trac/lcgdm/wiki/Dpm/Admin/Performance
[26] CreamTesting https://wiki.italiangrid.it/twiki/bin/view/CREAM/CreamTesting
[27] Workload Management System Testsuite https://wiki.italiangrid.it/twiki/bin/view/WMS/WmsTestSuite
[28] Robot Framework EMI Implementation
https://twiki.cern.ch/twiki/bin/view/EMI/RobotFrameworkQuickstartGuide
[29] LB deploy automation http://scientific.zcu.cz/scatter/scripts
[30] eInfrastructure for Testing, Integration and Configuration of Software (ETICS)
http://etics.web.cern.ch/etics/
[31] T-StoRM: a StoRM testing framework EGI Community Forum, Munchen 2012
[32] ShUnit testing http://shunit.sourceforge.net/
[33] Argus Functionality test https://twiki.cern.ch/twiki/bin/view/EMI/ArgusTestPlan
[34] Grinder load testing framework http://grinder.sourceforge.net/
[35] JUnit testing http://www.junit.org/
[36] Survey on Automated Testing in EMI
https://twiki.cern.ch/twiki/bin/view/EMI/EmiTestAvailabilitySurvey
[37] EMI Preview program https://twiki.cern.ch/twiki/bin/view/EMI/EMIReleasePreview
[38] Works with EMI program https://twiki.cern.ch/twiki/bin/view/EMI/EmiCollaborationPrograms
[39] EMI Large Scale acceptance testing https://twiki.cern.ch/twiki/bin/view/EMI/LargeScaleEMITestbed
[40] Future Grid project portal https://portal.futuregrid.org/
[41] PMD http://pmd.sourceforge.net/
[42] FindBugs http://findbugs.sourceforge.net/
[43] CheckStyle http://checkstyle.sourceforge.net/
[44] CCCC http://cccc.sourceforge.net/
[45] Pylint http://www.logilab.org/857
[46] CCppCheck http://cppcheck.sourceforge.net/
[47] Fedora Mock http://fedoraproject.org/wiki/Projects/Mock
[48] DejaGnu project http://www.gnu.org/software/dejagnu/
[49] Puppet http://puppetlabs.com/