Journal of Physics: Conference Series **119** (2008) 042010 doi:10.1088/1742-6596/119/4/042010

ETICS: the International Software Engineering Service for the Grid

Alberto Di Meglio[1], Marc-Elian Bégin[1], Peter Couvares[2], Elisabetta Ronchieri[3], Eva Takacs[4]

[1]CERN (Switzerland), [2]University of Wisconsin-Madison (USA), [3]INFN CNAF (Italy), [4]4D SOFT Ltd (Hungary)

Corresponding author e-mail: alberto.di.meglio@cern.ch

Abstract. The ETICS system is a distributed software configuration, build and test system designed to fulfil the needs of improving the quality, reliability and interoperability of distributed software in general and grid software in particular. The ETICS project is a consortium of five partners (CERN, INFN, Engineering Ingegneria Informatica, 4D Soft and the University of Wisconsin-Madison). The ETICS service consists of a build and test job execution system based on the Metronome software and an integrated set of web services and software engineering tools to design, maintain and control build and test scenarios. The ETICS system allows taking into account complex dependencies among applications and middleware components and provides a rich environment to perform static and dynamic analysis of the software and execute deployment, system and interoperability tests. This paper gives an overview of the system architecture and functionality set and then describes how the EC-funded EGEE, DILIGENT and OMII-Europe projects are using the software engineering services to build, validate and distribute their software. Finally a number of significant use and test cases will be described to show how ETICS can be used in particular to perform interoperability tests of grid middleware using the grid itself.

1. Introduction

Several large-scale open-source software projects have to deal with the need to organize complex software life cycle management infrastructures and processes in order to guarantee required levels of quality, interoperability and maintainability. Often these projects have to face resource, skill, time and budget constraints that may lead to the risk of releasing software difficult to deploy, maintain, understand and integrate with other applications. Research projects with limited duration have to focus on developing software of increasing functionality through their lifetime, but cannot always guarantee that the software will still be accessible, maintainable and documented after the conclusion of their mandate. If the projects are additionally geographically and administratively distributed across several organizations, ensuring that components developed by different developers, in different languages, on different platforms and with non homogeneous tools and processes is often a daunting challenge that may lead to software difficult to manage and maintain. Under the pressure of short deadlines and large requirement sets, project managers may have to face the decision of cutting testing and quality assurance verifications, which can ultimately cause delays in the releases or diminished usability of the software due to the excessive number of undetected problems.

Even when functional tests are performed, the nature itself of complex middleware, such as that developed for the grid, the provision of adequate hardware and network resources is a cost that no project but the largest ones can easily afford. When middleware and applications are deployed on tests

© 2008 IOP Publishing Ltd

or certification testbeds a lot of time is usually spent trying to make middleware suites and applications to interoperate due to the different configuration assumptions and different versions of common libraries.

In this work, we present ETICS [1], an integrated infrastructure for the automated build, configuration, integration and testing (BCIT) of software, discussing its architecture and presenting a number of examples of usage by large grid projects funded under the FP6 EC framework. ETICS aims to support research and development initiatives producing distributed and grid software by integrating existing procedures, tools and resources in a coherent infrastructure and providing an intuitive access point through a web portal and a professionally-managed, multi-platform resource facility based on Grid technologies.

ETICS doesn't replace configuration management tools like CVS [2] or SVN [3] or compilation tools like make [4] or ant [5]. Rather, ETICS provides a continuous integration and testing framework where such tools can be used. The advantage of using ETICS rather than the individual tools is that the specific syntax and usage requirements of each tool are abstracted by the ETICS data model allowing the software developers and maintainers to manage large, complex and distributed code bases with a unique, consistent set of commands and definitions. Other continuous integration and testing tools exists already (notable example are CruiseControl [6] and Maven [7]). However, most existing tools are dedicated to Java developers only and do not support execution of builds and tests 'jobs' on remote resources and local operations only are supported.

This paper is organized as follows. Section 2 describes the system architecture; Section 3 documents the data model, explaining how ETICS represents and uses information. Section 4 describes the different ETICS web applications, while Section 5 describes the command line tools and the APIs; Section 6 presents the Metronome job management system used by ETICS as execution engine; Section 7 outlines the Grid Quality Certification Model, a proposal to standardize the quality assessment of grid and distributed software projects; finally Sections 8 and 9 presents a number of real use cases and the future work foreseen.

2. ETICS Overview and System Architecture

The ETICS system is being developed by a project partially funded by the European Commission in the context of the FP6 program. It is composed of five partners, CERN (the project coordinator), INFN, Engineering SpA, 4D SOFT Ltd and the University of Wisconsin – Madison. The foreseen duration of the project is two years, from 1st January 2006 to 31st December 2007. An extension for two more years has been requested as part of the FP7 program.

ETICS aims to establish a distributed and managed infrastructure providing common software engineering tools and processes. Integrated pools of resources and easily accessible interfaces to the build and test tools have therefore to be provided and maintained for running automated builds and test suites. In addition all build and tests operations that can be executed on the remote worker nodes have to be reproducible locally on user computers to allow users to design and verify their testsuites and build configurations.

A centre of exchange of software configuration information and documentation is required in order to allow projects to organize and store metadata about their software and access information about other existing projects. A repository of standard tools, packages, documentation and interoperability information is required so that new and existing projects can easily and rapidly create new build configurations and testsuites.

Taking into account the above mentioned requirements, the ETICS system architecture is split into several entities, as shown on Figure 1: the ETICS Web Service; a number of web applications; a set of command line interfaces and APIs; the data model and its persistent storage implementation; and the job execution engine.

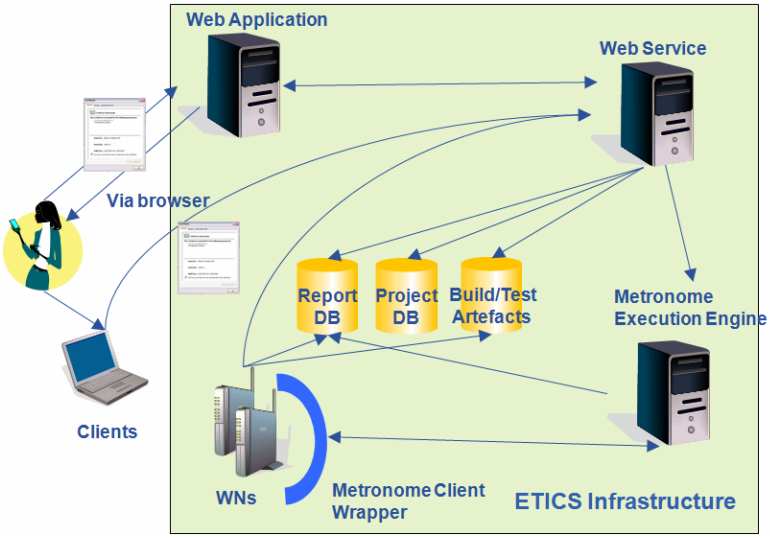

Figure 1: The ETICS System Architecture

2.1. System Features

The ETICS Build and Tests system comes with a default set of features to build software and test packages and services. The initial set of features can additionally be augmented by adding plugins [8] to the system providing specialized services. ETICS uses a plugin framework with a published interface that allows users or service providers to customize the system behavior as necessary.

The initial set of features comprises a job management engine to automate the execution of build and test jobs with support for a wide range of operating systems, CPU architectures and compilers. The internal data model allows specifying dependencies between packages or tests, which are automatically managed by the build and test engine and compiled or deployed as required. The basic set of functional plugins allows to trigger triggers coding convention checks, unit test execution, documentation generation and to collecting and publish the build and test results to the ETICS online repository.

The ETICS system supports several Version Control Systems (like CVS and SVN) and can be extended to additional systems. It has an internal packaging system that allows developers to generate automatically packages in various formats (e.g., tarballs, RPMS, debs, MSIs) and for different platforms by specifying simple sets of properties. The current version of the ETICS system (v. 2.0.3) supports several operating systems such as Scientific Linux (SLC3, SLC4, SL5), Ubuntu 7, Debian 4, various versions of Red Hat Linux, CentOS, MacOSX and Windows (on a smaller set of features) on both 32 and 64 bit (ia64 and x86_64) CPUs.

The build engine can be instructed to build large package sets entirely from source or as a combination of source and precompiled binaries and supports strict release processes by providing volatile package repository for the development phases and permanent, non modifiable repositories for release distributions, ensuring the reproducibility of the entire build and test process over time. The compilation architecture is normally automatically detected and the appropriate binaries are downloaded from the repository. However, it is also possible to override the automatic detection specifying an alternative platform in order to perform cross-platform builds. This can be useful for example to run 32-bit builds on 64-bit (x86_64) platforms (this requires some support from the modules to be built in order to accept the necessary compilation flags).

2.2. Authentication and Authorization

The underlying security infrastructure is based on digital certificates [1]. The web applications and the command line client authenticate themselves using standard x.509 [9] certificates. Users are modeled as fully qualified x.509 principal names as they appear in standard x.509-compliant certificates. The ETICS Web Service verifies the user certificate Distinguished Name in the database of existing users

involved in a project, and it allows or denies the operation based on the roles assigned to the users. From that point onwards, the Web Service uses a service certificate to interact with other internal services. The access control list on the persisted data is enforced by the Web Service. The following roles are defined in the system:

Role	**Description**
Administrator	A super user enabled to perform all the operations allowed in the ETICS infrastructure
Module Administrator	Responsible for handling individual projects
Developer	Works on the implementation of the software and can submit remote builds
Integrator	Runs remote builds and registers packages in the package repository
Tester	Runs remote tests stores test results in the test repository
Release Manager	Responsible for defining project level configurations and releases for a project
Guest	Read-only access to publicly accessible information

3. The Data Model

The data model and the storage are designed based on the principles of industry standard information models like CIM [10] and introduce formal entities to organize the software projects structure, the build configuration, the security information, the build and job result set and so on. The data model describes explicitly the objects and the relationships between objects (as in the CIM model). In addition, the model allows representing the results of running a build and test job in a way that can be consumed by the web application to generate reports.

A software project structure is represented using three main objects: the Project, the Subsystem and the Component. A Project can contain Subsystems or Components and a Subsystem can contain Components. Although the model also allows nested Subsystem, this is not currently implemented. The three structural objects (generically referred as Modules) represent respectively the abstract identity of a project, a subsystem or a test node and a physical package to be built or a test to be executed, but no version information. The Configuration object is instead used to represent the version information of a Module. Every module can have more than one Configuration. A build or test list is composed of the project structure and an associated list of configurations. Not all structural modules in a build or test list needs having an associated configuration, which is equivalent to building or testing subsets of the full structure.

Every Module can be associated to sets of Version Control System Commands, Build Commands, Test Commands, Properties, Environment Variables and Dependencies. The sets are associated to the modules via an intermediate Platform object, which allows specifying separate sets per platform. The Platform object normally represents a particular combination of Operating System, CPU architecture and compilers or interpreters.

Dependencies are used to specify that a configuration has to be built or tested before another one. Dependencies can be static if they constrain the relationship between two named Configurations or dynamic if the specify a relationship between a Configuration and a Module. In the latter case, the Configuration of the Module to be used during a build or test run is resolved at run-time using project-wide information. This method is used to control the dependencies at the project level and enforce consistency across a project.

Following the typical CIM format, the relationships between the various objects are described by relationship objects. For example to associate a set of versions and dependencies to a module ETICS uses:

- One 'component' object containing generic properties of the module (name, license, repository root URL, etc) and a Globally Unique ID (GUID)
- One or more 'configuration' objects containing the version information, version control tags, path on the repository, etc) and a Globally Unique ID (GUID)

- One 'componentConfiguration' object for each 'configuration' object associating the GUID of the module with each GUID of the configurations
- One 'vcsCommand', one 'buildCommand' and one 'testCommand' object per configuration and per platform containing the checkout commands, the build commands (e.g. init, configure, compile, package, publish, etc) and the test commands (e.g. init, configure, test, publish, etc) and the GUIDs
- One or more relationship object to associate each configuration with its commands
- If required, one or more 'property', 'environment' and 'dependency' objects per configuration and per platform (each one with its own GUID)
- One or more relationship object to associate each configuration with its properties, environment variables and dependencies

In addition, the dependencies can be declared 'static' or 'dynamic'. A static dependency is a relationship between two named configurations (e.g. 'webservice v. 1.0.0' depends on 'tomcat v. 5.0.24'). A dynamic dependency is a relationship between a named configuration and a module (e.g. 'webservice v. 1.0.0' depends on 'tomcat'). In the case of dynamic dependencies, the exact configuration to be used during the build is resolved by ETICS during the execution of the build using global project information (e.g. a project mandates that 'tomcat v. 5.0.24' must be used for all components depending on 'tomcat') or using version constraints (e.g. 'tomcat >= 5.0' is specified and the highest available version in the ETICS database satisfying the constraint is used).

The data storage back-end holds the persisted data model in the form of a relational DB implementation based on MySQL [11]. It supports different deployment models. For example, the Web Service, the Web Applications and the DB can be hosted on the same node, but as the number of requests increases with more users using the service, the database can be hosted on a separate node.

4. The Web Service and the Web Applications

4.1. The ETICS Web Service

The Web Service is the entity providing business logic for the entire system and it is used by the command line client and by all web applications. An important goal of the web service is to abstract the data storage backend, which holds the persisted version of the ETICS data model. For simplicity and better scalability, the web service is stateless. This means that it does not use a stateful web service paradigm, such as Web Services Resource Framework [12], which still has to prove itself in high-availability applications. The ETICS Web Service is developed in Java and runs as a service in a Tomcat [13] container. It provides a set of high level methods to read and write objects from and into the database and a set of specialized methods to generate object factories, manipulate objects or get information about the operations performed by users and the system for auditing and logging purposes. Additionally, the Web Service abstracts the access to the underlying job execution engine presenting a common build and test job interface layer for remote submission.

4.2. The Build and Test Web Application

The Build and Test Web Application[1] is responsible for providing a presentation layer to allow the user to browse, create and modify Modules and all associated objects. In addition it provides data forms to submit remote build and test jobs to be execute in the ETICS resource pools. It is a stateful application in order to maintain the security credentials and session information. It is developed in Java and runs as a service in a Tomcat container. The current version uses an open source, but proprietary framework called jduck [14] (developed by one of the ETICS partners, Engineering SpA), but a new version using the more standard Google Web Toolkit [15] is under preparation.

4.3. The ETICS Repository

[1] https://etics.cern.ch/eticsPortal

During the build and test process, both locally on user computers and remotely on the ETICS build nodes, the ETICS client assembles a report including information generated during the build or test such as the commands used to execute the job, high-level information about the module and configuration being built or tested, the execution logs generated during the process and customized reports generated by the metrics collectors or by tools activated during the execution (such as static metrics generators, unit tests engines, compliance checkers, etc). The build and test reports generated during builds and tests performed on the ETICS resource pools are stored in the centralized ETICS Repository and are accessible via the Repository web application[2]. Also in this case the application is developed as a Java web service to be run in a Tomcat.

The ETICS software repository is also the standard location where all the software artifacts are stored (e.g. packages, build and test reports). The repository is composed of two main parts: the volatile storage area and a permanent storage area.

The Volatile storage area contains temporary user-defined storage directories that users can allocate on the repository for their private use. Packages and documents in the Volatile storage areas are always overwritten with the most recent files produced at the end of a build or test run. Files in the volatile areas are deleted after 40 days to make space for new ones.

The permanent (i.e. Registered) storage area is the official location where to store officially released and registered packages and documents. A package is registered when it can be made public (e.g. final version of a configuration). Files in the permanent area are never deleted and never overwritten with newer versions, therefore only the first ever produced instance of a specific package or document is stored.

The ETICS repository is accessible using a Java web application that gives access to both the Volatile and the Registered storage areas of the ETICS repository. The current implementation provides read-only access to the repository and allows downloading packages and documents. A full-featured web application based on digital library concepts is under development and will provide secure read and write access and a set APIs to implement high-level repository solutions and to federate multiple repositories together.

4.4. The ETICS User Administration Application

Users and roles in ETICS can be managed using the ETICS Administration application[3], a Java web application for Tomcat. The application can be used to manage user certificate registration and to assign to each user (that is each certificate) one or more access roles. Roles are assigned per project, per module or per configuration and allow managing access to the ETICS system and operation with a very fine grained control.

5. The CLIs and APIs

The Command Line Interfaces (the ETICS Client) provide a similar functionality as the web application and makes use of the same web service interface for simplicity and symmetry. The client is developed in Python and can be used directly by the user on local resources (for example a developer machine) to execute the same build and test operations that are executed by the system in the internal, controlled build nodes. Reproducibility of the results is guaranteed by executing each build or test in independent, isolated workspace where the environment is fully managed by ETICS Client using the information stored in the central database and package repositories.

The client also provides functions to create and manage build and test configurations in the local workspace without using the information in the central database [16]. This functionality allows developers to model their software and try it in different conditions before committing the information in the central database.

6. The Metronome Execution Engine

[2] https://etics.cern.ch/eticsPortal/#repository
[3] https://etics.cern.ch/eticsPortal/#legAdmin (requires access privileges)

The Metronome Job Execution Engine [17] developed by the University of Wisconsin-Madison allows the ETICS system to fully automate the submission of builds and tests on distributed resource pools. Build and test jobs can be executed on a regular schedule, on a large set of different combinations of operating systems, compilers and CPU architectures. The Metronome engine relies on Condor, a specialized workload management system for computing intensive jobs.

Metronome is in practice used to schedule the deployment of an ETICS CLI client and a set of build or test instructions for the ETICS client to execute. The actual execution of the build or test on the grid worker node is performed by the ETICS client. All information about configurations and dependencies are therefore taken at run-time from the central ETICS database allowing the execution of distributed builds and tests/

The resource pools are currently managed by CERN, INFN and the University of Wisconsin-Madison and collectively provide for public use more than 120 physical build nodes and more than 30 distinct combinations of operating systems, compilers and CPU architectures. Individual projects using ETICS can additionally use dedicated private resources where the build and test jobs can be executed under strict control.

6.1. The Co-Scheduling Engine

The ETICS/Metronome system implements a mechanism to allocate multiple nodes to a test job so that different components of the test (services, clients, test suites, etc) can be executed on separate nodes [8]. This functionality, known as C-Scheduling, allows deploying complex multi-node tests and simulate the interaction of clients, services and applications in realistic situations.

A complex test suite can be modeled in ETICS as a set of nodes and each node can be assigned one more components from the ETICS repository. The components can be middleware services or applications, clients, test scripts and so on. Helper deployment modules are also available to deploy automatically standard services like Tomcat or MySQL.

When a co-scheduled test is submitted to the ETICS system for execution, the test structure as modeled in the ETICS database is analyzed and a job submission file is automatically generated and used to instruct Metronome to pre-allocate the required number of nodes. Once all nodes have been allocated, the test components are deployed on each node using the standard ETICS Client. The different services are started in random order and are then synchronized by using the internal information system. The ETICS client provides a set of APIs to advertise and retrieve properties among the deployed services. The properties can be used inside the deployment scripts or the test scripts as semaphores to synchronize the execution of each service. For example, Service A can initialize and then advertise its IP address, while Service B depending on the Service A starts its initialization and the waits (with a configurable timeout) for the IP address to be advertise before proceeding.

The different test components signal to the system their status and in particular when they have finished executing and are ready for clean up. The system keeps track of the status of each allocated node and once all components have finished executing, the results are sent back to the Metronome submission server and the nodes are cleaned restoring them to the initial conditions for accepting new jobs.

7. Examples of usage of ETICS

All the entities described in the data model can be edited by using the CLIs and the Web Application. In particular the CLIs allow users to script the entire build or test process using simple sets of commands. As an example we consider the use case of a developer who needs to create a new configuration (a version) of a module as part of a larger project, register the new configuration in the ETICS database, build it, edit the configuration to modify information like dependency types or versions and rebuild it. The following set of commands can be used to perform the describe operations:

Set the project to work with and get the project structure	*etics-get-project <project>*
Checkout an existing configuration of a module	*etics-checkout –c <configuration> <module>*
Build the configuration	*etics-build <module>*
Use the existing configuration to create a new configuration	*etics-configuration prepare –o configuration.ini <module>*
Edit the configuration file changing version number, cvs tag, dependencies, etc as needed	*Use any text editor*
Register the new configuration	*etics-configuration modify –i configuration.ini*
Checkout the new configuration to get any new dependency (use –merge to preserve existing information)	*etics-checkout –merge –c <configuration> <module>*
Rebuild	*etics-build –c <newconfiguration> <module>*
Submit a remote build on two different platforms and register the binaries in the ETICS repository	*etics-submit build –register –platforms <platform1,platform2> –c <newconfiguration> <module>*

The configuration.ini file uses a specific format based on the standard INI format to describe all module and configuration properties, the dependencies, the checkout, build and test commands and so on.

8. The Grid Quality Certification Model

One of the major goals of the ETICS project is to define a Quality Assurance certification model suitable for assessing and monitoring software quality for grid projects. During its first 18 months of operations, a model has been defined in collaboration with the grid projects using ETICS and proposed for feedback at a number of international events (recently at the EGEE 07 conference in Budapest). The model, which is currently called Grid Quality Certification Model (GQCM) for its main application area, is however suitable for assessing in general the quality of distributed software development projects [18].

The GQCM model position itself between standard QA guidelines like CMMi [19] or ISO/IEC 25000:2005[4] and individual software metrics definitions. The standard models normally define qualitative guidelines for an organization, from where the expected quality of the software product can be inferred, while individual metrics provide quantitative ways of assessing specific, but normally disconnected properties, which have to be considered in specific context to be given meaningful interpretations.

We believe that there is a currently a gap between the standards and the commonly used software metrics, which Grid Quality Certification proposes to fill by defining a practical implementation of the QA guidelines using specific set of metrics and assessment perspectives. In this respect, GQCM is closer to and compatible with ISO 9126[5], but provides the "how", not only the "what". GQCM specifies in particular predefined and coherent sets of metrics and thresholds and ways of interpreting and consolidating them, whereas ISO 9126 leaves to the software development organization to define them.

GQCM and its implementation in the ETICS system provide therefore homogeneous, not subjective and comparable results and a repeatable workflow, extremely valuable especially in contexts where the QA culture or expertise is not strong or too costly to acquire.

[4] 'Software engineering — Software product Quality Requirements and Evaluation (SQuaRE) — Guide to SQuaRE', http://webstore.iec.ch/preview/info_isoiec25000%7Bed1.0%7Den.pdf

[5] ISO (1991). *International Standard ISO/IEC 9126. Information technology -- Software product evaluation -- Quality characteristics and guidelines for their use*, International Organization for Standardization, International Electrotechnical Commission, Geneva.

GQCM is structured in families or Evaluation Modules (EM). Each EM groups a set of QA evaluation techniques in a specific QA area: Coding style, Static analysis, Structural testing, Functional testing and Standards compliance. Each EM is associated to a one or more metrics within the QA area, which QA systems implementing the model must collect using the tools of their choice. Additionally, the metrics results can be grouped in the three points of view or 'perspectives', Quality, Platforms and Standards.

The perspectives are independent, contrary to other waterfall models and provide a partial score that contributes to produce the overall QA score of the software under assessment. Individual projects are also free to focus the Quality perspective alone or in combination with another perspective in case for example multi-platform support or compliance to standards is not applicable or desired. The overall score allows monitoring the progress of the software over time, taking corrective actions in deficient areas and comparing the status of different software systems.

9. ETICS Build and Test System Application Examples

9.1. EGEE/gLite

The EGEE project[6], co-funded funded by the European Commission and coordinated by CERN, brings together experts from over 32 countries with the common aim of building on recent advances in Grid technology and developing a service Grid infrastructure which is available to scientists 24 hours-a-day. The project provides researchers in academia and industry with access to a production level Grid infrastructure, independent of their geographic location.

The EGEE project also produces a grid middleware distribution called gLite, which is born from the collaborative efforts of more than 80 people in 12 different academic and industrial research centers. gLite provides a bleeding-edge, best-of-breed framework for building grid applications.

The gLite development process has been standardized on the ETICS system for building the software and testing the deployment of the service metapackages (such as the User Interface, the Worker Node, the WMS Server and the CREAM Computing Element Server). The entire gLite distribution is composed of about 580 distinct packages and almost 3000 configurations for a total of more than 1.6 million lines of code in C/C++, Java, Python and other languages.

gLite					
Modules	579	Configurations	2932	Dependencies	11179
- Project	1	- Project	8	- Internals	6414
- Subsystem	39	- Subsystem	320	- Externals	4764
- Component	539	- Component	2604	- Others	1

Before moving to the ETICS system, the gLite distribution was only developed on a single platform (CERN Scientific Linux 3 32bit), while it is now automatically built on many different platforms several times per day (CERN Scientific Linux 3 32bit, CERN Scientific Linux 4 32bit and 64bit, Scientific Linux 5, Red Hat Enterprise Server 4, Debian 4, Ubuntu 7, CentOS 4). Some experimental builds on subsets of the gLite components have been performed on MacOSX (Leopard). More recently a new project called Grid4Win (coordinated by INFN in Italy and sponsored by Microsoft) has adopted ETICS as build system in the effort of porting some of the gLite components to Windows using the Unix Services for Windows on Windows Server 2008.

9.2. DILIGENT

The main objective of DILIGENT is to create an advanced testbed that will allow members of dynamic virtual e-Science organizations to access shared knowledge and to collaborate in a secure, coordinated, dynamic and cost-effective way [20]. This testbed has integrated Grid and Digital Library (DL) technologies, where regression test activity is performed. It has been validated by two

[6] EGEE Middleware Architecture, August 2004, https://edms.cern.ch/file7476451/1.0/Architecture.pdf

complementary real-life application scenarios: the former comes from the culture heritage domain, the latter from the environmental e-Science domain.

DILIGENT is a Java based system that uses the Web Service Core framework of the Globus Toolkit comprised of approximately 90 components, which are Web Services, libraries, port-lets, and wrappers of gLite components, developed by several teams across Europe.

Diligent					
Modules	581	Configurations	1688	Dependencies	5326
- Project	1	- Project	14	- Internals	5024
- Subsystem	27	- Subsystem	103	- Externals	302
- Component	553	- Component	1571	- Others	0

The DILIGENT project has started using ETICS for building and testing remotely through Metronome. The current testing environment of the DILIGENT system consists of six worker nodes (i.e., virtual machines). ETICS local and remote testing facilities assist the DILIGENT testing process by executing remote regression tests and applying the static analysis and metric calculation utilities - being included as 'plugins' - in the Test and Metrics Framework of ETICS in an automated and straightforward way.

9.3. OMII-Europe

The OMII-Europe project[7] aims to bring together the best technologies from Europe and elsewhere and make them available in a usable and supported form to scientists. It delivers production quality middleware relevant to large-scale and smaller collaborative Grids by federating development activities across Europe. It also maximizes benefits of global collaborations with USA and China. Specific actions performed by this project covers for example the assessment of the QA process, the re-engineering of the adaptations to standards, the setting up of a central repository for software, documentation and training.

OMII-Europe uses ETICS to build and test its re-engineered middleware components on different platforms, the same supported by the original providers, and to perform deployment tests of its services. In addition, this project has to check interoperability between job submission engines by validating their conformance to the OGSA-BES recommendation. In order to do so, test suites have been developed in order to interpret and parse the recommendation into specific service calls, together with independent clients to perform tests. ETICS has therefore been adopted to perform this test automatically as part of the QA practices. OMII-Europe has a dedicated ETICS installation at the University of Southampton and also uses the public distributed testbed.

9.4. EUChinaGrid/EGEE/ETICS IPv6 Collaboration

EUChina[8], EGEE and ETICS are collaborating on the testing of IPv6 compliance of a number of component in the EGEE/gLite distribution. Each partner in this collaboration focuses on a specific task: EUChina has defined IPv6 code compliance tools and metrics to perform static compliance analysys; EGEE has worked on the porting of some gLite components to IPv6 in both pure IPv6 and dual stack configurations; finally ETICS has contributed providing the tools and resources to automated the IPv6 tests across three different sites (CERN in Switzerland, GARR in Italy and IN2P3 in Paris) and to produce full compliance reports of all software registered in the system [8].

10. Conclusions and Future Work

[7] OMII-Europe, http://omii-europe.org/OMII-Europe
[8] EUChina Grid, http://roc.euchinagrid.org/home.php

In this paper, we presented ETICS, a system that provides services and tools for automatic building, configuration and testing of distributed and grid software. Then, we provided an overview of the ETICS architecture split into several entities like the ETICS Web Service going in details for each of them. We also detailed one of the major goals of the ETICS project that consists of defining a Quality Assurance certification model for grid and distributed software development projects. Finally, we described examples of ETICS build and test applications, such as EGEE, DILIGENT, OMII-Europe and EUChinaGrid, also involved during the system development.

We believe that the ETICS system has potentialities in solving many of the problems faced by open-source software projects during their life cycle management processes, in particular guaranteeing high levels of quality, interoperability and maintainability. Therefore, we consider this project only the beginning of a large one. We intend to add extra plugins to the ETICS system in order not only to evaluate quality and interoperability metrics but also to perform complex tests like the installation of services. We also plan to improve the current ETICS performances considering users' feedback and to add further features accordingly to the needs of the ETICS community.

References

[1] Bégin M-E, Diez-Andino Sancho G, Di Meglio A, Ferro E, Ronchieri E, Selmi M and Zurek M 2007 *Springer Verlag Lecture Notes in Computer Science (LNCS) Series, LNCS 4401* pp 81-97
[2] Purdy G N 2000 *CVS Pocket Reference* (O'Reilly)
[3] Mason M 2005 *Pragmatic Version Control Using Subversion [Illustrated] (Paperback)* (Pragmatic Bookshelf)
[4] Talbott S 1991 *Managing Projects with Make* (2nd Ed. O'Reilly & Associates, Inc.)
[5] Hatcher E and Loughran S 2002 *Java Development with Ant [Illustrated] (Paperback)* (Manning Publications)
[6] Trueman T 2004 *Cruise Control (Hardcover)* (HarperTeen)
[7] Massol V and O'Brien T 2005 *Maven: A Developer's Notebook (Developer's Notebooks) [ILLUSTRATED] (Paperback)* (O'Reilly Media, Inc.)
[8] Bégin M-E, Couvares P, Diez-Andino Sancho G, Da Ronco S, Di Meglio A, Dini L, Fabriani P, Gietz B, Pavlos A, Ronchieri E, Selmi M, Takacs E and Zurek M 2007 *Tenth World Conference on Integrated Design & Process Technology* (Antalya, Turkey)
[9] Gutmann P 2000 *X.509 Style Guide* (http://www.cs.auckland.ac.nz/~pgut001/pubs/x509guide.txt)
[10] McMorran A, Ault G, Morgan C, Elders I and McDonald J 2006 *Power Engineering Society General Meeting 2006* (IEEE)
[11] DuBois P 2005 *MySQL (Third Edition)* (Sams Developer's Library)
[12] Foster I, Frey J, Graham S, Tuecke S, Czajkowski K, Ferguson D, Leymann F, Nally M, Storey T, and Weerawaranna S 2004 *Globus Alliance*
[13] Brittain J and Darwin I F 2003 Tomcat*: The Definitive Guide [ILLUSTRATED] (Paperback)* (O'Reilly Media, Inc.)
[14] Godfrey M and Grossman D 1999 *Thirtieth SIGCSE technical symposium on Computer science education*
[15] Chaganti P 2007 *Google Web Toolkit: GWT Java Ajax Programming (Paperback)* (Packt Publishing)
[16] Bégin M-E, Da Ronco S, Diez-Andino Sancho G, Gentilini M, Ronchieri E and Selmi M 2007 *International Conference on Computing in High Energy and Nuclear Physics(CHEP)*
[17] Pavlo A, Couvares P, Gietzel R, Karp A, Alderman I D, Livny M and Bacon C 2006 *LISA06: Twentieth Systems Administration Conference* (Washington DC, USA) pp 263-273
[18] Rippa A, Bégin M-E, Di Meglio A and Manieri A 2007 *Third Conference of the EELA Project*
[19] Chrissis M B, Konrad M and Shrum S 2003 *CMMI – Guidelines for Process Integration and Product Improvement (Hardcover)* (Addison-Wesley Professional)
[20] Castelli D, Candela L, Pagano P and Simi M 2005 *2nd IEEE – CS International Symposium Global Data Interoperability* (IEEE Computer Society) pp 56-99