ETICS Meta-data Software Editing - From Check Out To Commit Operations

Marc-Elian Bégin[1], Saverio Da Ronco[2], Guillermo Diez-Andino Sancho[1], Mattia Gentilini[3],Elisabetta Ronchieri[3], Matteo Selmi[3]

[1] CERN, CH-1211, Geneva 23, Switzerland
[2] INFN, University of Padova, Via Marzolo 8, I-35131 Padova, Italy
[3] INFN-CNAF, Via Ranzani 12/2, I-40126 Bologna, Italy

E-mail: elisabetta.ronchieri@cnaf.infn.it

Abstract. People involved in modular projects need to improve the build software process, planning the correct execution order and detecting circular dependencies. The lack of suitable tools may cause delays in the development, deployment and maintenance of the software.

Experience in such projects has shown that the use of version control and build systems is not able to support the development of the software efficiently, due to the large number of errors that cause the breaking of the build process. Error common causes are for example the adoption of new libraries, libraries incompatibility, the extension of the current project in order to support new software modules.

In this paper, we describe a possible solution implemented in ETICS, an integrated infrastructure for the automated configuration, build and test of Grid and distributed software. ETICS has defined meta-data software abstractions, from which it is possible to download, build and test software projects, setting for instance dependencies, environment variables and properties. Furthermore, the meta-data information is managed by ETICS reflecting the version control system philosophy, thanks to the existence of a meta-data repository and the handling of a list of operations, such as check out and commit. Because of this, all the information related to a specific software are stored in the repository only when they are considered to be correct.

By adopting this solution, we show a reduction of errors at build time. Moreover, by introducing this functionality, ETICS will be a *version control system like* for the management of the meta-data.

1. Introduction

Software developed and maintained in Grid environments [1] has to face problems of common accesses to the same resources by different people. They often are geographically spread around the world and are not able to communicate each other. For these reasons, users adopt Version Control Systems (VCSs) to maintain their code, allowing to save the same files without any agreement. VCS is very efficient when handling single file, but is not useful when people have to work with the meta-data, which often need to express relationship between elements. Users may profit a lot from the possibility of having local copies of the meta-data without connecting and saving them into the database.

Software needs to be built and tested, therefore users adopt several well-known tools [2], such as Makefile, autotool, ant for making software more portable and to simplify building it. In general, these tools are not valid when software project is deeply complex containing a lot of

dependencies, in particular when it is required to build on several platforms and automatically to schedule daily and nightly build, and when it needs to handle different artifact and report repositories.

In this paper, a brief introduction to the most common VCSs such as SubVersion (SVN) [3] and Concurrent Version System (CVS) [4; 5] is provided, explaining how they solve the problems of synchronization and how they handle the possibility for a developer to maintain local copies of the files. A comparison amongst a production system and two VCSs is given in order to provide an evaluation of similarities and differences. The software meta-data editing in ETICS is described, detailing basic software concepts and some operations generally used by users. Finally, several possible workable scenarios in the ETICS system are shown.

ETICS is a software infrastructure, developed for the configuration, build and test of software. Currently, it is already used by some important European Grid projects such as EGEE [6], Diligent [7] and OMII-Europe (see *OMII Europe* at `http://omii-europe.org/OMII-Europe/`). Analyzing some use cases provided by real cases experienced by Grid projects, such as EGEE, we derive a number of requirements a system like ETICS has to support in order to provide a local editing solution. In addition, describing these use cases, we explain how the Grid software can benefit from the implementation of these requirements.

The paper is structured as follows: Section 2 introduces related work. We introduce the basic concepts to model meta-data of a generic software project in Section 3. Section 4 highlights the architectural framework for the application of software meta-data editing modelization. Then, Section 5 discusses some use cases, whilst Section 6 describes the implementation of the software meta-data editing in the ETICS system. Section 7 details three scenarios of the ETICS usage. Finally, Section 8 concludes the paper.

2. Related Works

Usually, meta-data are handled automatically, as a result of the different actions triggered by the users. These actions will be in general executed in a storage system (typically a database) where the user has no access to the meta-data. However, the idea of duplicating part of the meta-data information locally, might result extremely useful, though it does not seem to have been used so far. The typical work-flow that uses a VCS is performed as follows: the most current version of code is checked out in the local work area by the developer; then, local changes are performed to local code; finally, after tests and analysis of results from the code run are deemed satisfactory, changes are committed back to the repository. The same working procedure is provided by ETICS to handle meta-data. In what follows, we give a brief review of the most popular VCSs, known as SVN and CVS.

2.1. CVS

CVS is probably the most popular VCS. Like all the version control software, CVS provides a way to maintain information about project evolution in order to retrieve prior versions of files, track changes and coordinate the efforts of a developers team. In the context of VCSs, the *repository* is the centralized area that stores the project's files. The CVS repository contains information required to reconstruct previous versions of the files in a project, while a work area contains copies of version of files from the repository. New development occurs in work areas, and any number of work areas can be created from a single repository. Work areas are independent of each other and may contain files from different development stages of the same project. The CVS has all the characteristics exposed previously, moreover it follows a *merging model* to handle possible conflicts. This model allows everyone to have access to the files at all times and supports concurrent development. The model is in fact optimistic: it assumes that conflicts are uncommon and that when they do occur, *usually* it is not difficult to resolve them. CVS tracks file versions by a revision number, which can be used to retrieve a particular

version from repository. In addition, CVS supports the so called *tagging* procedure, that is the possibility to create symbolic tags so that groups of files can be referred by a single identifier. Finally CVS supports the branching development. Even if the typical development schema is linear, sometimes there is the need to parallelize the development of code. This can be done by creating a branch from an appropriate point of the development of the project. If and when the two branches need to be re-united a merging procedure is provided.

2.2. SVN

SVN is a more recent VCS, born with the aim to overcome CVS, eliminating some problems that are typical in CVS. Anyway SVN has its own difficulties, and today it is a concurrent package more suitable than CVS in certain development environments, but less performant in other contexts. It embraces, like CVS, the merging model, therefore it is optimistic. The repository format in the SVN case is based on relation database, whilst in CVS is based on revision control system files of version controls. Adopting a database allows SVN to use the transaction mechanism (not present in CVS), therefore the operations commitment and checkout can be either completed or failed in toto and not partially. A possible workable scenario consists of having a user who checks in several files, transferring them to the server. It could happen that the operation performed by the user is only completed for some of these files, and not for the rest due to conflict reason for example. In this case, the transaction mechanism is able to perform a roll-back operation, restoring the state available before the check in. Moreover, the database structure of SVN turns into the quickest time response of the SVN commands with respect to CVS. Again, SVN adopts a different numbering convention to store file revisions: SVN takes a unique enumeration for the whole project, whilst CVS uses a different number revision for each file. In addition, even though SVN allows transactions, it does not support the roll-back operation after a wrong commitment, whilst CVS does. Clearly, restoring procedure can be performed from the repository side, even if this is not the best way to solve the issue.

3. Description of Software Meta-data Editing

In this section we describe the basic concepts used by ETICS to model meta-data for a generic software project, and the operations required by users.

3.1. Basic Concepts

First, a software project is mapped to the following ETICS meta-data: *component* that is a portion of code, *subsystem* that is a container of components, and *project* that is a container for components and subsystems. The general term *module* is then used for referring to project, subsystem and component. Each module has associated at least one configuration, which is meta-data containing information for checking out, building, testing and handling of software dependencies. Configurations can be linked each other in order to produce a tree of configurations that starts from a project configuration to components configurations. Another ETICS meta-data contains platform information, such as operating system, architecture and compiler version. Each configuration contains platform specific data related to commands, environment variables and software dependencies. Commands are used to build, checkout and test software.

3.2. Operations

Then, considering user and a given work area, we describe the operations to perform in this context. They allow users to interact with a database in order to get meta-data information and to update them after having applied some changes in a local work area.

Checkout a module *This operation gets meta-data information about the selected modules and configurations.*

Add a module *This operation adds a module.*

Add a configuration *This operation adds a module configuration*

Clone a configuration *This operation clones an existing configuration*

Modify a module *This operation replaces some information about module.*

Modify a configuration *This operation modifies the configuration information such as checkout, build and test commands, software dependencies or environment variables.*

Remove a module *This operation removes a module.*

Remove a configuration *This operation removes a configuration.*

Commit *This operation updates information of modules and configurations.*

We notice that a roll-back mechanism can be defined when a list of operations is performed. When all the operations have a successful result, they are performed. Whilst if at least one of them has an error result, none of them is performed (i.e., the project is untouched).

4. Architectural Framework of Editing Strategy in ETICS

To understand the environment for which the ETICS system has been developed, an overview on its architectural framework is here given. ETICS is an infrastructure for Configuration, Build and Test (CBT) of software, based on the requirements coming from distributed software projects [8]. It also addresses portability issues and interoperability testing [9]. It is organized into four layers, as shown in Figure 1: (1) client layer; (2) business layer; (3) scheduler layer; (4) data layer.

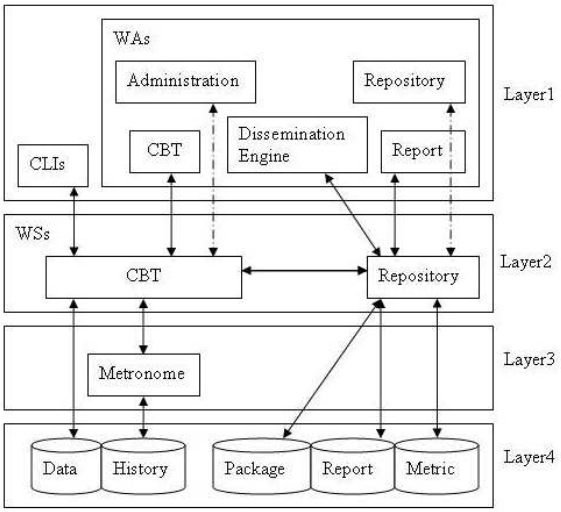

Figure 1. ETICS Architectural Framework

The core of the system is the central CBT Web Service (WS). This is the main entity at the business layer, and it is intended to provide logic for the entire service, handling meta-data information, user requests, and contacting the execution engine to commit build and test jobs. The client part of the system, acting at the first layer, is mainly provided by a Command Line Interface (CLI). This interface is able to send requests to CBT WS, to collect meta-data in

a local workspace, to edit them and finally is able to use this information to perform build and test jobs. Because of these features and its high portability, the CLI is used not only by developers to edit meta-data, but also by the remote nodes in order to execute test and build jobs. It is worth to notice that a Web Application (WA) is also provided by ETICS to edit meta-data. Anyway even if this interface is more *user friendly* than a standard CLI, it is less powerful, because it is neither able to allow local editing nor able to execute build or test jobs. To complete the simplified sketch of the ETICS architecture, it is important to mention that ETICS provides a scheduler layer (the third in Figure 1) based on the Metronome software [10]. Metronome allows CBT WS to offer the user the automation of builds and tests - possibly on a regular schedule - on a large set of different resources and platforms. ETICS provides also a set of tools to implement the publication of information collected during the execution of test and build jobs, such as the Report, the Dissemination engine and the browsable repository, all accessible via a WA Interface. In addition, ETICS implements secure access to meta-data, providing Administration interfaces and a secure access model based on digital certificates that enforce the x509 encryption standard [11].

5. Use Cases

Many scenarios in the configuration of software registered in ETICS can benefit from a "local" management solution. Therefore, software developers, working for different distributed software projects and using ETICS to test, integrate and configure them, asked to have this possibility. In what follows, some projects that expressed interest in the local editing solution are presented. Moreover a description of their use cases is given in order to define the ETICS implementation.

5.1. Projects demanding local editing solution

gLite [12] is a lightweight Grid computing middleware developed by EGEE, which is a collaborative project aimed at building a seamless Grid computing infrastructure for e-Science. gLite is the software that enforces the EGEE Grid project specification, enabling users to access the computing infrastructure capabilities. It is currently deployed on hundreds of sites that participate in EGEE to serve many disciplines, notably experiments running on the LHC accelerator (see *LHC-THE LARGE HADRON COLLIDER* at `http://lhc.web.cern.ch/lhc/`) , such as ALICE, ATLAS, CMS and LHCb.

OMII-Europe is an European-funded project aimed at sourcing key software components that can inter-operate across several heterogeneous Grid middleware platforms, endorsing use of both open standards and open source software. The OMII-Europe task is to develop quality-assured Grid services running on existing major Grid infrastructures, such as EGEE, putting emphasis on re-engineering of software components rather than on development of new technology. Its targets are inter-operability and quality assurance. It aims at establishing itself as an impartial broker, giving advice on heterogeneous Grid solutions. OMII-Europe currently involves a set of sixteen established partners from Europe, USA and China, and middleware platforms such gLite and Globus [13].

DILIGENT has the main goal to create an advanced test-bed for the creation of digital libraries that support collaboration between virtual organisations. DILIGENT also aims at disseminating Grid technology, proposing a cost-effective digital library model, and promoting cross-fertilization between the two domains.

5.2. Description of use cases

Use cases defined below have been applied to the configuration and module elements defined in the ETICS system. These elements represent a sub-set of the meta-data software abstractions. The analyzed operations are *add*, *modify*, *prepare*, *remove*, *clone* and *rename*, as described in Table 1. They change meta-data, using a plain-text representation of their state.

Table 1. The analyzed editing operations

add	creates a new data
clone	copies existing data (used for configurations only)
modify	modifies existing data
rename	changes the name of existing data
prepare	generates a plain-text representation of existing data
remove	deletes existing data

A user may modify a dependency, for example adopting an upgraded module, and try to build a module in order to verify the correctness of the code before committing dependency changes. In addition, a user may add a new configuration in order to verify a new module version in coming of a new release. A user may also clone an existing configuration in order to change only few parameters, for instance required during a checkout. Then, another use case is the deletion of a module in order to test the re-structuring of the project to simplify it. A user may create a representation either for new meta-data which will then be added to the project, or for existing meta-data (e.g., the default configuration of new modules) using a file in order to save it in one of the VCSs adopted by the project.

6. Implementation in ETICS

In the ETICS system, the handling of meta-data related to software project configuration is a fundamental task, and ETICS developers addressed to it with particular care, keeping in mind the needs of the users and their requests about more freedom in the development and test phases of the project growth. In the world of code management, the data handling of VCSs such as SVN or CVS, inspired the ETICS developers to use the technique of local work area to manage meta-data. The solution adopted by the ETICS client is a sort of mix between the two VCSs. The main difference with CVS and SVN is that ETICS works with meta-data and not with files. Meta-data are stored in a centralized relational database, but for editing purposes they are stored temporarily in a *XML* document in the work areas. ETICS does not implement a revision number: even if the database takes track of previous versions of meta-data, these can be retrieved just using the time stamp. Moreover, whilst the VCSs just provide a work area where developer uses its own tools, ETICS also provides the commands to handle information stored locally, keeping track of changes performed using an history file, like CVS. This simplified solution is anyway powerful in the meta-data context. Finally, as SVN does, ETICS provides the transaction mechanism during the commitment, avoiding the corruption of meta-data stored in the central database, and also provides a support for the roll-back procedure.

Before the development of the local editing solution, any single modification done in a project registered in the ETICS infrastructure, required a connection to the ETICS WS. This improvement introduces a "transaction" mechanism, similar to the one present in many Database Management System, in which many modifications are done locally and a single server connection is done to apply all the operations. By adopting this solution, atomicity is guaranteed: i.e., if an operation fails, all the operations already done will be rolled back, leaving the ETICS database in the state it was before the transaction. Users can choose whether to run commands with this behaviour or not. Users have a work area in which the project information gathered from the ETICS WS are stored; when using the local editing solution, the work area also tracks the history of modifications made to the project. Removing a meta-data deletes all history references to it, if the history contains the creation of that meta-data; otherwise, the modification operations

are removed. In any case, a failure in an operation leaves the local work area and the operations history as they were before that operation.

Figure 2 shows the order of the actions performed by the local and non-local solutions. If a user adopts a non-local solution, first the editing operation interacts with the ETICS Web Service, second with the user work area. In the case of a local solution, first the editing operation interacts with the user work area, second with the ETICS Web Service by using the command *etics-commit*.

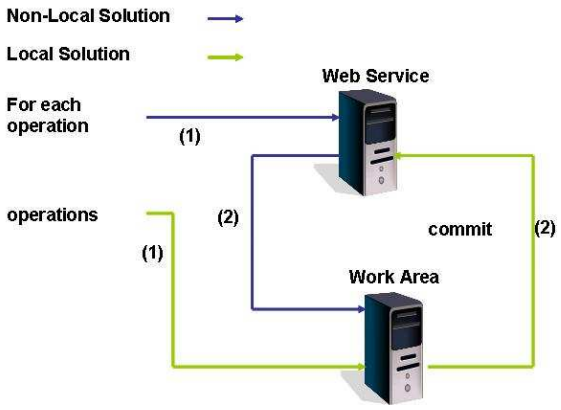

Figure 2. Local and Non-local solutions actions. (1) and (2) represent the order of the actions.

ETICS provides various commands to handle the meta-data (some editing commands are described on Table 2.

Table 2. Few editing commands

etics-module	used for project, component and subsystem data, generically called module
etics-configuration	used for configuration
etics-commit	used for committing changes performed locally

When a local editing command is executed, all the local changes are tracked in a local history file. Different commands, such as *etics-module* and *etics-configuration* support different operations, as described in Table 1. In order to simplify the meta-data representation we decided to adopt *ini format* because it is friendly for end-users.

7. Scenarios

7.1. Case I

A possible scenario consists of a user that downloads a project, called P. Then, the user creates a subsystem A that contains two components, called A1 and A2. Next, the user modifies subsystem and project configurations in order to link respectively the new component configurations (i.e., A1.HEAD, A2.HEAD) and subsystem configuration (i.e., A.HEAD). Finally, the user commits changes performed locally.

Example of Component-A1.ini file:

```
[Component-A1]
licenceType=ETICS
vendor=ETICS
description=None
repository=http://eticssoft.web.cern.ch/eticssoft/repository
download=None
packageName=A.1
homepage=None
vcsroot=:pserver:anonymous@etics.cvs.cern.ch:/cvs/etics

[Parent]
Subsystem=A
```

It is important to observe that the ini file name contains as prefix one of the following values: Component, Subsystem, Project and Configuration. The prefix is followed by plus a name.

7.2. Case II

Another scenario consists of a user that downloads a project, called P1. Then, the user modifies an existing component configuration (B.HEAD) changing a dependency, then adds a new component configuration (C.HEAD), next modifies the previous configuration (B.HEAD) adding the new one as dependency. At this point the user commits changes performed locally.

Example of Configuration-B.HEAD.ini file:

```
[Configuration-B.HEAD]
profile=None
status=None
moduleName=P1.B
description=P1.B v. 1.0.0
version=1.0.0
path=None
age=1
tag=B_branch_1_0_0

[Platform-default:VcsCommand]
checkout=echo hi

[Platform-default:StaticDependency]
externals|globus=globus 3.2.1,B
P1|C = C.HEAD,R
```

7.3. Case III

In this scenario, a user downloads a project. Then, he adds a new component to the project, modifies its configuration, performs a checkout using the local meta-data, builds the new component and commits the new meta-data. A possible sequence of commands is shown in the following list:

```
etics-module prepare --component dmctk
etics-module add --noautocommit -i Component-dmctk.ini
etics-configuration prepare -c dmctk.HEAD dmctk

etics-configuration modify --noautocommit -i Configuration-dmctk.HEAD.ini
```

```
etics-checkout --local -c dmctk.HEAD dmctk
etics-build dmctk
etics-commit
```

8. Conclusions

In summary, we detailed how the local editing solution can improve the programming activities of developers. We formalized the solution describing the basic concepts and the operations involved in it. The ETICS architecture was explained focusing on the editing functionality. We described some use cases, addressing the needs of some Grid projects, such as EGEE, OMII-Europe and DILIGENT. We compared the ETICS solution with what is provided by the most common VCSs, such as CVS and SVN. Finally, we described three workable scenarios.

Acknowledgments

We would like to thank colleagues from the ETICS project, who have provided useful suggestions to complete this paper. This work is partially funded by the European Commission under contract number INFSOM-RI-026753.

References

[1] Foster I and Kesselman C 2003 *The Grid 2: Blueprint for a New Computing Infrastructure* (San Francisco, CA, USA.: Morgan Kaufmann Publishers Inc.)

[2] Gianelle A, Pelusoy R, Sgaravatto M, Giacomini F, Ronchieri E, Avellino G, Cantalupo B, Beco S, Maraschini A, Pacini F, Guarise A, Piro R, Werbrouck A, Kouril D, Krenek A, Kabela Z, Matyska L, Mulac M, Pospisil J, Ruda M, Salvet Z, Sitera J, Vocu Mand Mezzadri M, Prelzx F, Monforte S, Pappalardo M and Colling D 2004 *CHEP 2004* vol 2

[3] Mason M 2005 *Pragmatic Version Control Using Subversion [Illustrated] (Paperback)* (Pragmatic Bookshelf)

[4] Purdy G N 2000 *CVS Pocket Reference* (O'Reilly)

[5] Berliner B 1990 *Proceedings of the USENIX Winter 1990 Technical Conference* (Berkeley, CA: USENIX Association) pp 341–352 URL `citeseer.ist.psu.edu/berliner90cvs.html`

[6] Gagliardi F 2005 *Lecture Notes in Computer Science* **3402** 194–203

[7] Castelli D, Candela L, Pagano P and Simi M 2005 *2nd IEEE - CS International Symposium Global Data Interoperability* (IEEE Computer Society) pp 56–59

[8] Bégin M, 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* 81–97

[9] Bégin M, 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)

[10] 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

[11] Housley R, Ford W, Polk W and Solo D 2002 Internet x.509 public key infrastructure: Certificate and crl profile updated by RFC 4325,RFC 4630

[12] Di Meglio A, Flammer J, Harakaly R, Zurek M and Ronchieri E 2004 *Computing in High Energy and Nuclear Physics (CHEP)* vol 1 (Interlaken, Switzerland) pp 579–582

[13] Foster I 2006 *Springer-Verlag LNCS 3779* IFIP International Conference on Network and Parallel Computing