

EDOS: Environment for the Development and Distribution of Open Source Software

Serge Abiteboul, Xavier Leroy, Boris Vrdoljak
INRIA, France
{serge.abiteboul,xavier.leroy,boris.vrdoljak}@inria.fr

Roberto Di Cosmo
Paris 7 University, France
roberto@dicosmo.org

Stéphane Fermigier
Nuxeo, France
sf@nuxeo.com

Stéphane Laurière, Frédéric Lepied, Radu Pop,
Florent Villard
Mandrakesoft, France
{slauriere,flepied,rpop,warly}@mandrakesoft.com

Jean-Paul Smets
Nexedi, France
jp@nexedi.com

Ciarán Bryce
University of Geneva, Switzerland
ciaran.bryce@unige.ch

Klaus R. Dittrich
Zurich University, Switzerland
dittrich@ifi.unizh.ch

Tova Milo, Assaf Sagi, Yotam Shtossel
Tel Aviv University, Israel
milo@cs.tau.ac.il

Eleonora Panto
CSP Torino,
Italy
eleonora.panto@csp.it

Abstract – The open-source software community is now comprised of a very large and growing number of contributors and users. The GNU/Linux operating system for instance has an estimated 18 million users worldwide and its contributing developers can be counted by thousands. The critical mass of contributors taking part in various open-source projects has helped to ensure high quality for open source software. However, despite the achievements of the open-source software industry, there are issues in the production of large scale open-source software (OSS) such as the GNU/Linux operating system that have to be addressed as the numbers of users, of contributors, and of available applications grow. EDOS is a European project supported by IST started October 2004 and ending in 2007, whose objective is to provide a new generation of methodologies, theoretical models, technical tools and quality models specifically tailored to OSS engineering and to software distribution over the Internet.

I. EDOS PROJECT'S MOTIVATION

To understand the emerging problems related to Linux distributions engineering, it is necessary to reconsider the process of producing such software. Each version of a GNU/Linux operating system is known as a distribution. The code of a distribution is composed of a large number of modules, written by contributing developers who are generally independent and spread out over different countries. It is the role of the distribution editor to collect the modules and to integrate them into a new distribution with respect to their inter-dependencies. This process is known as packaging. It involves extensive testing and module correcting. Once the distribution is ready, it is put at users' disposal through a network of mirrors and peer servers. Further roles of the distribution editor are to develop customised versions of a distribution for specific clients to make versions available to users in an efficient manner.

Figures related to the case of Mandrakelinux distribution give an order of magnitude of the whole process: the code base of Mandrakelinux release contains 7000 object files and over 3000 source files. Along with information for dependencies and documentation, the total image distributed in a release is around 20 GBytes. The total effort required by Mandrakesoft is around 30 person-years to produce each new distribution release, at a pace of two new releases per year.

The goal of the EDOS project [1] is to improve two main aspects of the distribution process: (i) packaging and testing, and (ii) code distribution. The goal is to dramatically increase the engineering productivity. To achieve this goal, EDOS project has the ambition to innovate all along the production chain of a GNU/Linux distribution.

II. FORMAL MANAGEMENT OF SOFTWARE DEPENDENCIES

The effort needed in producing distributions is partly due to the complexity in managing dependencies between the large amount of modules, or packages, that make up the distribution.

The problem is twofold: on one side, editors need to keep up with recent source code changes by developers, which is a manual and error-prone task, which often leads developers and editors to backtrack to cater for interim changes to modules.

On the other side, a distribution editor must ensure that, when a set of packages is rubberstamped as a stable distribution, then it is consistent, and allows successful installation of each reasonable user selection of packages out of this set. Even better, the user expects, when moving from an old version of a distribution to a newer one, to find an upgrade path that does not disrupt her system.

To tackle this problem, it is necessary to properly handle dependencies among packages and among features of packages, like configuration or compilation options.

In cases where dependencies are incorrectly handled, inconsistent versions of the system are produced which simply do not work or compile on end-user machines.

The sheer size of a modern distribution makes automated support and verification of dependencies a necessity.

EDOS addresses these issues by issuing a formal component dependency description model, as well as providing tools to perform static analysis on software repositories.

III. EDOS TESTING FRAMEWORK AND QUALITY ASSURANCE PORTAL

The second angle of work in the EDOS project addresses the testing issue. In the context of OSS, testing is not only functional unit testing, but also regression testing (ensuring backward compatibility) as well as coverage testing (ensuring that the tests are sufficiently complete).

EDOS provides a unified, computer technology-neutral testing format and leverages this format by setting up an online testing platform helping contributors to have an accurate provable view of the system they are building together, and to report test results. Testing a GNU/Linux operating system, or indeed any large application built on OSS, is a time-consuming but essential operation.

The EDOS testing framework allows tests and their results to be “outsourced” in a way similar to how source code development is outsourced. New tests can be added by developers, by the distribution editor as well as by theoreticians in the same manner that the developers add new modules. This approach imposes new security requirements. As the number of developers becomes larger, it is indeed important to be able to introduce accountability and to allow developers to securely attach meta data to source files so that one can trust the test data associated with the files.

A first version of EDOS testing framework was released in March 2005 in the form of `umitest` [2], a file format and a promising prototype implementation. It implements the concept of Automated system Integration Based quality assurance which is a novel way to test the stability of GNU/Linux distributions by simulating a complete rebuild of a collection of hosts and running test scripts on each host to verify functionalities. It is based on the reuse of the numerous existing unit testing, functional testing and package testing tools designed by open-source software communities. `umitest` provides the initial core engine for the quality assurance portal which is under implementation. It is already used in production for automating unit tests and functional tests of a Mandrakelinux based Live CD and of an open source Enterprise Resource Planning system.

Mandrakesoft has developed `Testzilla` [3], a collaborative testing framework connected to `Bugzilla`. `Testzilla` defines simple procedures such as: "insert USB key, wait 5 seconds, check if removable disk is mounted on a desktop and asks users to run the procedures and report success or failure to a central database". `Testzilla` has been extended to implement experimental automated testing. A simple XML file defines which packages need to be installed on the top of a base system and which scripts should be run on the resulting system. A farm of PCs is

automatically reinstalled thanks to the Mandrakelinux autoinstall tool. Simple tests are run automatically. Results are shared in the Testzilla central database, with bugs directly passed to the common Bugzilla.

Fig. 1 below provides an illustration of the integrated approach.

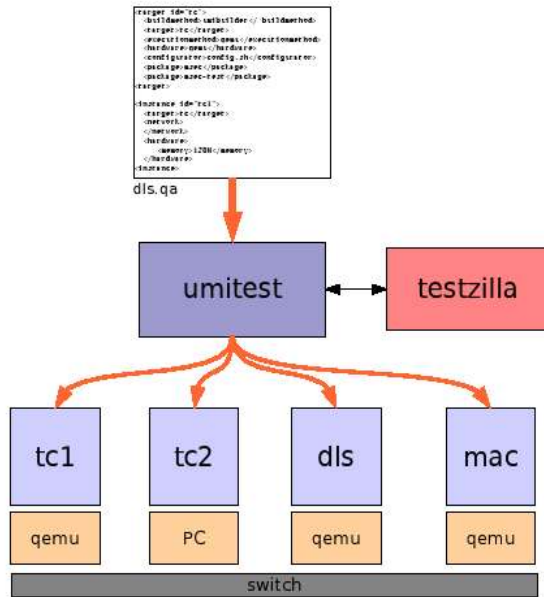


Fig. 1. Automation of system integration tests overview

The file `dls_cd.qa` represents for example a system test involving one thin client server (dls), two thin clients (tc1 and tc2) and one Macintosh computer (mac). It consists of verifying that each thin client can connect to the thin client server and that the macintosh can access files on the thin client server. Tc1, dls and mac are executed on virtual machines (qemu). Tc2 is executed on a real PC. The system image of tc1, tc2 and dls are built with umibuilder. The system image of mac is provided as a read only disk image. Configuration is achieved on each image by using a special copyonwrite image and copying some files generated by a umiboot configuration script.

Once all systems are set up, umitest executes shell scripts on each host through the console access provided by the host. Using the console access provided a guarantee that all parts of the system can be tested, including the boot part, can be tested automatically. In our approach, complex testing is achieved by building two virtual hosts: one which achieves the tests and the other with is the tested host. For example, one virtual host could run function / performance tests on another virtual host. Triggerring tests on the first virtual host does not need more than ability to access the console an run a script. Results of tests are published in Testzilla.

EDOS team has provided a complete working prototype which was tested on real world problems. This prototype is for now integrated in the umigumi open source project and

can be used immediately.

Based on the current prototype, future directions have been listed:

- 1) The current testing framework implementation should be extended to better support real hosts as in Testzilla experimentation. This requires combining umibuilder with autoinstall in Mandrakelinux.
- 2) Ability to gather test results and associate them to a bug tracking system is needed, as in Testzilla for example, or in a collaborative QA portal managed by workflows.
- 3) Ability to generate complex scenarii could be added to the system design so that the creation of a complex network of hundred hosts can be automated rather than described.
- 4) Ability to track code quality in source RPMs though the addition of a .sqa file format which described code tests and generates a code metric.

Future extensions are expected with the goal to refactor the current experimentation in a more general framework: the Quality Assurance Portal.

IV. NOVEL DISTRIBUTION OF CODE OVER THE INTERNET

The third and last technical angle in the EDOS project addresses the issue of the distribution of the new releases to end-users and developers. A P2P architecture will be designed, tested and compared to more classical distributed database architectures based on replicas and mirroring. The goal is to support a very high quality and very efficient transmission of code bases from editor to end-user. A P2P system is one that does not require a centralised control. It is made up of a relatively large number of members who both contribute to, and profit from, membership of the community. In the context of EDOS, the aim of the P2P system is to store copies of the software release. The members of the open-source community are the members of the P2P system, a sufficient number of members for a P2P system for this approach to be feasible. The built-in parallelism of a P2P system and the increase in the number of systems who help distribute the software will bring numerous positive effects compared to the current master-slave architecture, and in particular :

- 1) Reliability and accessibility: because of the replication in the P2P system, the process would not rely on the correct functioning of any particular server for the distribution to be made.
- 2) Performance would increase with the number of

peers available to satisfy a given request. In particular, a user will be able to download code from several peers in parallel, and will also be able to take advantage of the proximity of some copy.

3) The time required for a modification to propagate through to the end-user will be dramatically reduced.

4) Finally, the system will allow peers to customise some distribution and provide it to selected communities in push or pull mode.

From the perspective of data sharing and namely code distribution in a P2P environment, the KadoP system [4,5] (Knowledge and Data in Peer to Peer) is getting adapted. KadoP relies on distributed hash tables technology, XML indexing and query optimisation techniques, and on the paradigm of ActiveXML [6] documents, to enable the publication and efficient large-scale querying of XML-centred content in a P2P environment. KadoP takes advantage of intentional XML documents to dynamically compose data-driven web services. ActiveXML is used at the application level for describing the content of the peers and the desired exchange of information and software fragments between peers: in ActiveXML documents, some of the data is given explicitly and some is given only intentionally by means of calls to web services.

V. EDOS METRICS

EDOS transverse measurement effort consists in defining and instrumenting indicators that describe the production process, the characteristics of the community involved in the process, the quality of the final product and the improvements occurring between release cycles. Inspiration is drawn from existing quality models and proposed metrics are specifically tailored to OSS engineering issues.

EDOS methodology leans on the evaluation process described in the ISO/IEC 14598 standard [7], the process of definition of quality models described in the ISO/IEC 9126-1 [8], as well as on some elements of the Goal/Question/Metric method [9].

The methodology consists of the following steps:

- 1) Identifying the purpose and goals of the measurement and evaluation
- 2) Specifying a quality model
- 3) Defining metrics
- 4) Establishing a measurement and evaluation plan
- 5) Data collection - measurement, rating, evaluation
- 6) Interpretation

In respect to the distribution of OSS for instance, the purpose of the measurement and evaluation is to compare different architectures used for the distribution.

In order to be effective, specific and explicitly stated goals should be specified. Each goal has to be focused on a certain aspect of the code distribution process. As suggested by the ISO/IEC 14598 standard, the evaluation process can be described from different points of view, such as the point of view of editors, developer users or end users. Developer users are those who contribute to the development of the software by testing the current software version and reporting bugs. The end users are those who are primarily interested in getting a stable version of the software.

Following goals have been identified for our purpose:

- 1) Improvement of the quality of service from the point of view of developer users
- 2) Minimisation of the costs from the editor's point of view.

After explicitly stating the goals, quality models with corresponding metrics are to be defined using ISO/IEC 9126-1 standard for software product quality as a starting point and an inspiring example. Still, we have to keep in mind that EDOS requires specific quality models, since the characteristics to be measured relate to the code distribution process, rather than just the software quality itself. It is fundamental to the preparation of any evaluation that a quality model reflecting the user's requirements of the objects to be evaluated is constructed. Therefore, we will focus to the first goal, and create a quality model for the quality of service in the code distribution process from the point of view of the developer user.

A quality model consists of a set of quality characteristics, each of which can be decomposed into a set of quality sub-characteristics. The structure is hierarchical, and, theoretically of unlimited depth. For the specified goal, i.e. improvement of the quality of service from the point of view of the developer user, the EDOS quality model consists in following criteria hierarchy:

Quality of content

Consistency

Freshness

Edit distance from the latest version

Time distance from the latest version

Ease of use

Time to get a package

Resource availability

Resource size

Transfer speed

Effort to get a package
Finding the resource
Choosing a server

User's cost

Space
CPU usage
Connection time

User's satisfaction

While the meaning of some characteristics is quite obvious from their name, other characteristics like consistency and freshness need a more detailed explanation. Consistency represents the edit distance between the set of packages on the target computer and the reference set of packages as it existed at the source server in the beginning of downloading. Regarding freshness, two different aspects have to be distinguished: the first one relates to the edit distance between the distribution version on a target server (or user's client) and the version on the main server. The second one deals with the time interval needed for arrival of a package (or the complete distribution version) after publishing on the main server. It is useful to make a distinction between freshness and consistency since many clients may run old versions of the OSS distribution. Freshness influences the implementation of the distribution network. For instance, if few users actually possess recent packages, then this impedes on the efficiency of a P2P based architecture.

Next stages in the EDOS metrics effort will consist in instrumenting further the designed quality model, in extending it to other aspects of OSS, such as dependencies measurement and eventually in providing a unified quality model specifically tailored to the artefacts of open-source software development, diffusion, usage and maintenance.

VI. POTENTIAL IMPACT ON THE OPEN-SOURCE COMMUNITIES

The EDOS project is about improving the process quality in producing OSS, and as a direct result, the quality of the software delivered to end-users. The whole project is fully committed to the open-source development and distribution model. As such, all the software deliverables as well as the intermediate revisions are freely accessible to the public. A web page is dedicated to the project, with links and software to download, including reports and articles. Beyond the lifetime of this project, the source code will live the life of regular OSS. Volunteers may pick up the job and carry on, any time, or embed part of it in their own projects.

As for the end-user perspective of OSS, two specific user communities will be involved. They will act as a

validation test-bed to provide feedback to the quality assurance portal prototype:

- 1) The Dschola Community [10], composed of secondary Italian schools, aiming to support the use of ICT in education. The Dschola community is concerned with using open-source in schools and promote it through different actions, like workshops, training and technical support for teachers.
- 2) The RIUSA Project devoted to old personal computers recycling using open-source software.

VII. REFERENCES

- [1] <http://www.edos-project.org>
- [2] <http://cvs.erp5.org/cgi-bin/viewcvs.cgi/umigumi/>
- [3] <http://qa.mandrakesoft.com/twiki/bin/view/Main/TestZilla>
- [4] <http://www-rocq.inria.fr/gemo/Gemo/Projects/KadoP/>
- [5] KadoP: Serge Abiteboul, Ioana Manolescu, Nicoleta Preda. "Constructing and querying peer-to-peer warehouses of XML resources". *International "Semantic Web and Databases" workshop (in cooperation with the VLDB conference)* Toronto, Canada, August 2004 (ps)
- [6] <http://activexml.net/>
<http://forge.objectweb.org/projects/activexml/>
- [7] ISO/IEC 14598-1:1999 *Information technology - Software project evaluation - Part 1: General Overview*, Geneva, International Organization for Standardization and International Electrotechnical Commission, 1999.
- [8] ISO/IEC 9126-1:2001 *Software engineering - product quality - Part 1: Quality model*, Geneva, International Organization for Standardization and International Electrotechnical Commission, 2001
- [9] R.Solingen, E.Berghout: *The Goal/Question/Metric Method*, McGraw-Hill, 1999.
- [10] E. Pantò, E. Lavagno, *Dschola - A regional school network*, TIEC Proceedings, 2002
<http://web.udg.es/tiec/posters/cp15.pdf>