Experiences with moving to open source standards for building and packaging

D H van Dok, M Sallé and O A Koeroo

Nikhef, Science Park 105, 1098 XG, Amsterdam The Netherlands

E-mail: `dennisvd@nikhef.nl msalle@nikhef.nl okoeroo@nikhef.nl`

Abstract. The LCMAPS family of grid security middleware was developed during a series of European grid projects from 2001 until 2013. Since 2009 we actively started to move away from ETICS, the project-specific build system, to common open-source tools for building and packaging, such as the GNU Autotools and the Fedora and Debian tool set. By following the guidelines of these mainstream distributions, and improving the source code to fit in with the commonly available open source tools, we have established low-cost, long term sustainability of the code base.

1. History

In 2001 the DataGrid project kicked off a series of European Commission funded projects to develop a science grid in Europe, complete with the software for resource management and grid services. Nikhef, together with the University of Amsterdam at the time, contributed several middleware security components: the LCAS/LCMAPS framework for site access control and user identity mapping, and gLExec for identity switching based on the above. Continuous development spanned more than a decade (from DataGrid through EGEE-I, II and III, with EMI and IGE ending in 2013). We maintain the software to this date.

Most of the middleware in these grid projects has been licensed under an open source license. (Currently, the Apache License, Version 2.0 is the most common choice.) At the time the DataGrid project got started, the term *open source* had been well-established and the practice of sharing source code was a long-standing tradition in the scientific field.

The 1990s saw the rise of the open source phenomenon, with flagship projects like the Linux kernel and the Apache web server. The pressure of commercial competition started a debate about the relative merits of opening source code to public scrutiny, and it led to many publications on the economical, sociological and security aspects of open source.

The question arises whether the much-trumpeted advantages of open source have improved the quality of grid middleware. A common theme in the success stories of open source projects is the building of an engaged community. Since grid middleware occupies a niche market it does not have a massive user base. Therefore, the choice for an open source license probably was not motivated by the mantra that 'enough eyeballs make all bugs shallow,' (Linus' Law, [1]), but rather by fitting the nature of such publicly funded work.

Over the years, the source code base grew. The following EGEE projects (I, II and III) saw a growth in partnership and contributions. As components were interdependent, the complexity of maintaining the code base and delivering the software to the growing number of participating sites in the European grid demanded the adoption of a central build system to manage it all. The earlier, home-grown system based on `ant` was abandoned in favor of ETICS [2, 3], which came out of EGEE in 2006 and would become a full-blown project in its own right.

While ETICS worked reasonably well, it incurred a regime of rigidly defined dependencies on exact versions of every piece of software (including those external to the project) in a hierarchy of projects, systems, subsystems and components which proved quite a management chore for everyone. As effort was spent making the components work with ETICS it became impractical to build anything outside it, or for outsiders to reproduce the build conditions of any of the components. This raised concerns about long term sustainability and adoption outside the original community, for instance by the Open Science Grid.

Since the project focused on producing software specifically for a single supported platform, Scientific Linux, there was as good as no building and testing going on for other platforms. Although the middleware used GNU Autoconf to figure out dependencies, the project used tailored `m4` macros that would work poorly or not at all outside the operating conditions set by ETICS.

With the third and last EGEE term drawing to a close, we started to worry about the future of our own contributions. If we wanted to maintain it ourselves, we needed to make sure that every aspect of the engineering and support surrounding the source code would be under our control.

2. Taking control

The first problem we needed to address was building the software and its dependencies (VOMS [4, 5], Gridsite [6] and Globus Toolkit [7]) outside of the ETICS environment. As the software uses the ubiquitous GNU Autotools [8], this was largely a matter of improving the dependency discovery code in the `m4` macros.

Much work went into the creation of a large shell script to build the software from scratch on any platform, but gradually we trimmed down the special cases covered in the 1700+ lines of this script. Our ultimate goal was to match the common pattern of just needing

```
./configure
make
make install
```

to build. A few notes about this general recipe must be made.

- The `configure` command should succeed if and only if all the prerequisites are met, i.e. all the required software is available on the system.
- If some required software is missing, the `configure` command must fail in a predictable manner.
- Once the `configure` command succeeds, the compilation must not fail.
- If required software is installed in an alternative location, it should be possible to pass this location to `configure` via command-line arguments.
- The installation must be redirectable to a staging area by way of setting the `DESTDIR` variable.

The Autoconf manual encourages building outside the source tree. This allows simultaneous builds for different architectures or different configuration settings. It takes some additional effort on the part of the developers to make this work, but the result is much cleaner. The ultimate workflow would look something like this:

```
tar xfz glexec-0.9.8.tar.gz
cd glexec-0.9.8
```

```
mkdir build
cd build
../configure
make
make distcheck
```

The `distcheck` target tests building outside the source tree, as well as clean `install` and `uninstall` targets. It then generates a source tar ball that can be distributed. This is the preferred way of generating distributable sources; it does not include version control artifacts or anything the developer may accidentally have left behind in the development source tree.

2.1. Code improvements

In the process of reaching the goal of being able to do this for all our middleware packages we encountered many bugs. Actively trying our distribution tarballs on a variety of platforms unveiled type mismatches and POSIX violations that went undetected as long as we stuck to the Scientific Linux platform.

One important choice we made was to stick to the lowest common denominator for our dependencies. Developers have a natural desire to make use of newly available features as they come out, but it may take a long time before a particular version of any piece of software trickles down to the main-stream distributions. Production grids typically still run either Scientific Linux 5 or CentOS 5, which came out in 2007 and will remain supported until 2017. We have always been careful about choosing features from the oldest supported distributions.

We wrote `man` pages for all programs and file types where they were missing, and generally updated the documentation.

Following an independent security audit, we refactored some code to adhere to more secure coding standards and removed unsafe constructs.

We improved the logging code and integrated logging to `syslog`.

2.2. Version control

The version control system used in the DataGrid/EGEE era was CVS, hosted by CERN. We obtained a dump of the CVS tree for the middleware we maintained and imported it in our own SVN repository. If we would not already have had an SVN server, we might have considered a distributed version control system such as Git, Mercurial or Bazaar.

2.3. Bringing out releases

Now that the software was in shape to produce distributable tar balls, we considered what would be the right way to create and distribute these. If users of our software report bugs or other problems related to a specific version, it is important that we know exactly which version of the sources their binaries corresponds to. We therefore implemented the following measures:

tagged releases Every time we prepare a release, we create an `svn` tag to mark the sources.

signed releases From these tagged sources, a distribution tar ball is created. The developer then takes SHA1 and SHA256 hashes from the tarball, and copies them in a GPG signed e-mail to the developer's mailing list.

permanent URLs The tar ball and hashes are uploaded to a web server which will be the canonical permanent location. The name of this server is chosen to be rather generic, so the URLs look like this:

    http://software.nikhef.nl/security/lcmaps/lcmaps-1.6.1.tar.gz

The exact steps are written down in our software procedures [9].

2.4. Packaging
Our responsibilities do not end with releasing sources. The goal of the EMI and IGE projects was to produce packages that fitted in with the local operating environment, and that meant producing `RPM`s for Red Hat Enterprise Linux based systems (such as Scientific Linux and CentOS) and `deb` packages for Debian and Ubuntu.

Creating a basic package is simple, but meeting the rigorous guidelines of both the Fedora [10] and Debian [11] distributions forced us to actually take this a step further.

The use of GNU Autotools turned out to be a real advantage, as this is considered as the standard build system in both Fedora and Debian and no special arrangements needed to be made in packaging.

Some care had to be taken in specifying build dependencies. It is recommended to use the available tooling for building in a clean-room environment, where only the minimal build dependencies are met. This helps to detect missing dependencies and ensures binary linkage using the correct paths. For Fedora, the tool to use is called `mock`. Debian packagers may consider `pbuilder` or `sbuild`. These systems produce standardised reproducible builds.

During the EMI project, ETICS was updated to do `mock` and `pbuilder` builds as well. For us this change proved to be a real simplification as we now could just build from our own source `RPM`s.

2.5. Automation
The packaging code is used for several distributions. The Fedora packaging is used for the latest Fedora releases (which come out semi-yearly), and the backports for the current Red Hat based systems, EPEL5 and EPEL6. The Debian packaging is used for Debian unstable, stable and oldstable and could easily be used for Ubuntu as well with minimal changes.

For automating the packaging builds, we have set up Koji [12] (developed and used by Fedora). By implementing a trigger in our Subversion repository, we initiate a series of package builds when a tag is created on the `SPEC` file for a package. Koji issues `mock` builds internally.

We have not found a suitable automation service for Debian. We have considered several systems, but all of them had insurmountable drawbacks. Debian's `buildd` is geared towards building a single, complete distribution, Canonical's Launchpad actively discourages running a personal version and the Open Build Service did not handle the matching of build time dependencies in the correct way. So far we have settled on using scripts to launch various builds with `cowpoke` and `cowbuilder` [13] (a variant of `pbuilder`).

The `RPM`s coming out of Koji are signed with a dedicated GPG key using `sigul` and compiled into a repository using `mash`. These tools are developed by Fedora and integrate with Koji.

The `deb` packages are signed with the developer's GPG key, and placed in a local repository with the help of the `reprepro` tool. The repository metadata is automatically signed with another dedicated GPG key.

The resulting repositories are hosted on `software.nikhef.nl`, and can be readily installed on target systems using standard package management tools such as `yum` and `apt-get`.

3. Results
The question arises whether the advantages gained by these adaptations are worth the effort which was needed. This is hard to measure objectively, so we can only present indicators from personal experience. It is relevant to note that we were already operating a medium-sized grid computing facility, and we were able to reuse existing systems for mailing lists, version control and virtual machine management for building and testing.

Moving to POSIX and ANSI C was a gradual process; we encountered a few standard violations and had to rewrite code occasionally. This helped the portability to other platforms, such as Debian and Mac OS X, and the transition from `i386` to `x86_64`.

Fixing the Autotools setup was considerable more effort, but it paid back in ease of porting, building and packaging.

Since the Open Science Grid is also using gLExec and LCMAPS, they benefited from the improved Autotools setup. Later they would turn to directly using our source `RPM`s.

The packaging for Fedora and Debian made up another large portion of work, but the available guidelines and documentation are excellent. Tools such as `rpmlint` and `lintian` helped to find packaging bugs. Although the guidelines are large bodies of text, they are all sensible and represent the distilled experience of hundreds of packagers.

The main benefit of the build and packaging automation is the reduced time to release a fix for any found bug. When a bug is reported, we should know exactly which sources correspond to it, so it is easier to reproduce. After fixing, rolling out a new release usually takes less than an hour altogether.

We aim for inclusion in Debian and Fedora, and have already gained from the experience of veteran packagers in these projects to improve our packaging code.

Staying close to common open source software proved useful when trying to solve the more obscure issues. Searching on-line for the answers often turned up identical findings on public mailing lists or forums. Being a member of a large user collective is certainly helpful.

4. Conclusions

Over the course of the last five years we have implemented many changes in the way we treat our software, by adhering to best practices in open source software development. In our personal experience these changes have proved to be worth the effort needed to achieve them. We can now operate in a state of low-cost maintenance. With the reduced amount of available manpower we are still able to address bugs and issues, and regularly bring out new releases.

Users of our software benefit from easier configuration and installation. Overall we believe that we have achieved long-term sustainability.

Acknowledgments

This work is part of the research program of the Foundation for Fundamental Research on Matter (FOM) which is financially supported by the Netherlands Organisation for Scientific Research (NWO).

This work is part of the activities of the Dutch e-Infrastructure, which is financially supported by the *Nederlandse Organisatie voor Wetenschappelijk Onderzoek* (Netherlands Organisation for Scientific Research, NWO) and the Dutch higher education and research partnership for network services and information and communication technology (SURF).

References

[1] Raymond E S 1999 *Knowl., Technol. & Policy* **12** 23–49 ISSN 0897–1986
url: `http://dx.doi.org/10.1007/s12130-999-1026-0`

[2] Di Meglio A, Bégin M E, Couvares P, Ronchieri E and Takacs E 2008 *J. Phys.: Conf. Ser.* **119** 042010
url: `http://stacks.iop.org/1742-6596/119/i=4/a=042010`

[3] Bégin M E, Ronco S D, Sancho G D A, Gentilini M, Ronchieri E and Selmi M 2008 *J. Phys.: Conf. Ser.* **119** 042004
url: `http://stacks.iop.org/1742-6596/119/i=4/a=042004`

[4] Alfieri R, Cecchini R, Ciaschini V, dell'Agnello L, Frohner A, Gianoli A, Lőrentey K and Spataro F 2004 *Grid Computing* (*Lecture Notes in Computer Science* vol 2970) ed Fernández Rivera F, Bubak M, Gómez Tato A and Doallo R (Springer Berlin Heidelberg) pp 33–40 ISBN 978-3-540-21048-1
url: `http://dx.doi.org/10.1007/978-3-540-24689-3_5`

[5] Alfieri R, Cecchini R, Ciaschini V, dell'Agnello L, Frohner A, Lőrentey K and Spataro F 2005 *Future Gener. Comput. Syst.* **21** 549 – 558 ISSN 0167-739X
url: `http://www.sciencedirect.com/science/article/pii/S0167739X04001682`

[6] McNab A 2005 *Softw.: Pract. and Exp.* **35** 827–834 ISSN 1097-024X
url: `http://dx.doi.org/10.1002/spe.690`

[7] Foster I and Kesselman C 1997 *Int. J. of High Perform. Comput. Appl.* **11** 115–128 (*Preprint* `http://hpc.sagepub.com/content/11/2/115.full.pdf+html`)
url: `http://hpc.sagepub.com/content/11/2/115.abstract`

[8] The Free Software Foundation Inc. 2012 *GNU autoconf – creating automatic configuration scripts*
url: `http://www.gnu.org/software/autoconf/manual/`

[9] van Dok D H and Sallé M 2013 *Site access control software procedures*
url: `http://wiki.nikhef.nl/grid/SAC_software_procedures`

[10] Callaway T *et al.* 2012 *Fedora project packaging guidelines*
url: `http://fedoraproject.org/wiki/Packaging:Guidelines`

[11] Jackson I *et al.* 2013 *Debian policy manual*
url: `http://www.debian.org/doc/debian-policy/`

[12] McLean M *et al.* 2013 *Koji - rpm building and tracking system*
url: `http://fedorahosted.org/koji/`

[13] The Debian Project 2012 *Cowbuilder tutorial*
url: `http://wiki.debian.org/cowbuilder`