SOFTWARE RELEASES MANAGEMENT IN THE TRIGGER AND DATA ACQUISITION OF ATLAS EXPERIMENT

Integration, Building, Deployment, Patching

Andrei Kazarov
CERN, Geneva, Switzerland, on leave from Petersburg Nuclear Physics Institute, Gatchina, Russian Federation

Mihai Caprini
National Institute of Physics and Nuclear Engineering, Bucharest, Romania

Igor Soloviev
Department of Physics, University of California Irvine, Irvine, U.S.A.

Reiner Hauser
Michigan State University, East Lansing, U.S.A.

Keywords: Software, Release, Package, Building, Maintenance, Patching, CMT, RPM.

Abstract: ATLAS is a general-purpose experiment in high-energy physics at Large Hadron Collider at CERN. ATLAS Trigger and Data Acquisition (TDAQ) system is a distributed computing system which is responsible for transferring and filtering the physics data from the experiment to mass-storage. TDAQ software is developed since 1998 by a team of few dozens developers. It is used for integration of all ATLAS subsystem participating in data-taking, providing framework and API for building the s/w pieces of TDAQ system. It is currently composed of more then 200 s/w packages which are available for ATLAS users in form of regular software releases. The s/w is available for development on a shared filesystem, on test beds and it is deployed to the ATLAS pit where it is used for data-taking. The paper describes the working model, the policies and the tools which are used by s/w developers and s/w librarians in order to develop, release, deploy and maintain the TDAQ s/w for the long period of development, commissioning and running the TDAQ system. In particular, the patching and distribution model based on RPM packaging is discussed, which is important for the s/w which is maintained for a long period on the running production system.

1 INTRODUCTION

ATLAS Trigger and Data Acquisition (TDAQ) system (Atlas, 2003) is a distributed computing system which is responsible for transferring and filtering the physics data from the ATLAS experiment to mass-storage. TDAQ project was started in the middle of 1990's as a number of small R&D projects. In 2010 TDAQ software is composed of more then 200 s/w packages which are built for 2 Linux platforms. The following table gives the overview of the s/w evolution in last 10 years in terms of number of packages and supported h/w platforms. The total number of source files in the s/w is currently more then 10000.

Table 1: TDAQ s/w evolution.

year	s/w version	N of packages	Platforms
2000	0.0.9	18	SunOs, LynxOs, Linux
2002	0.0.17	27	SunOs, LynxOs, Linux
2005	1.4.0	136	Linux SLC3
2010	2.0.3	210	32 and 64 bit Linux SLC4 and 5, MacOs (partially)

The TDAQ s/w is providing framework and API used by other ATLAS groups to develop s/w pieces which are all together compose the s/w running the ATLAS trigger and data-acquisition system at the experiment pit. Therefore, the proper s/w release development, deployment and maintenance policy is important to guarantee the smooth evolution of the s/w from the integration stage to the deployment and patching on the running system.

The s/w is made available to users in forms of regular s/w releases. The requirements for the s/w development model and s/w releases are listed below:

– support of h/w platforms and compilers used in ATLAS
– support of debugging
– support for different programming languages and s/w tool-kits: C/C++, Java, Python, IDL, Qt
– support of external s/w, integration with ATLAS-wide and CERN-wide s/w
– support for remote installation
– support for patching

The development environment provided to users should support the model of developing a package or set of packages against a particular release.

Releases must be installed on a shared file system (AFS, managed by CERN IT) for the testing and development, made available for download for the remote institutes and test labs and finally be installed for the use at ATLAS pit for data-taking.

Before being released, a stage of integration and testing of the s/w is necessary to guarantee the quality and functionality of the s/w.

Still being in the development stage, the s/w was actively used for integration with ATLAS subdetectors and for commissioning the system at the test beams (Gadomski, 2006).

To conclude, the s/w release model and tools must provide the functionality and flexibility which allow the development of TDAQ s/w, its testing and validation, the deployment for remote labs and institutes, the integration with the ATLAS experiment production s/w, and finally the maintenance of the s/w through the years of the life of the experiment.

2 S/W RELEASE MODEL

2.1 Definitions

Package: an independently developed piece of s/w providing some well-defined functionality in form of libraries, applications and data files. Normally there may be few developers contributing to a package. Typically the source code of a Package resides in a separate area in a code repository (e.g. SVN, see Section 3.1).

Version Tag: a reference in the code repository which uniquely defines some version of a package. Usually it has a form of <package>-MM-VV-PP.

Platform: a combination of h/w and s/w tags which identifies the type of binaries to be built. For instance: i686, Linux slc5, gcc43, debug, profiled. A release is build for a number of platforms (Section 3.2.2).

Release: a set of tagged packages built together for a number of platforms, providing some well defined and documented functionality for end users and made available for distribution. It is a way in which all efforts of developers of many packages is exposed to users in a common structure.

2.2 Release Policy and Live Cycle

The release policy should provide a good balance between moving the s/w forward and keeping it stable for end users. Two types of releases are foreseen by the model: major releases and minor releases. Major releases may contain important changes in the architecture of s/w, new functionality and API changes in packages, database schema changes and other similar changes which require some actions from end users.

Minor releases may contain some internal changes which do not require code changes at user's side.

In addition, a patching schema is foreseen, where a particular problem may be fixed by a binary patch to a package in the release which is already deployed to the system.

In the last years when the major development was done, 2-3 major releases per year were produced, and each one may be followed by 1-2 minor releases. In the present condition, when the release is used for data-taking in ATLAS pit, no major changes in s/w is possible, and all maintenance and implementation of new required features is made via the patching mechanism which is described in more details in Section 4.

On Figure 1 the life cycle of a typical TDAQ release is presented.

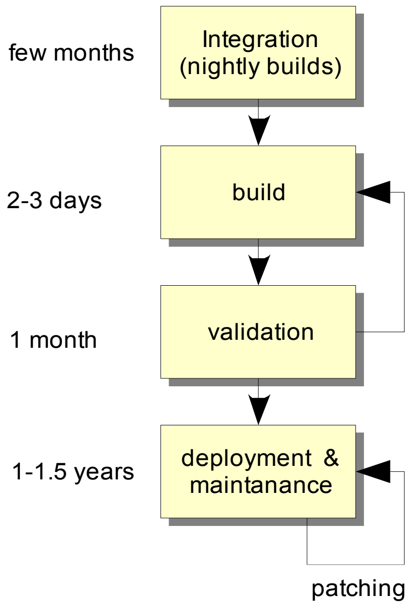


Figure 1: s/w release cycle.

The integration phase is finished when all the required functionality is available and all packages are successfully built altogether the nightly build (Section 2.4). Then the release is built and made available for testing on the shared filesystem and on the dedicated labs for the validation phase. In case of major problem found in this phase, the release may be rebuilt including new tags of packages which failed in the validation.

After validation phase, the release made available for download and can be deployed to the production sites. From this point, the patching procedures are activated (Section 4.1.2).

2.3 Scope of TDAQ s/w, External Dependencies

2.3.1 Scope and Projects

TDAQ s/w is organized in a tree of "projects" or sub-releases: *tdaq-common*, *dqm-common* and main *tdaq* release. Such factorization made possible the integration of TDAQ s/w with other ATLAS s/w projects as shown in Figure 2.

Such integration allows to use many commonly used packages provided by other projects (described in the next section) but from other side it dictates the platforms and compilers which can be used in the project and also directly affects the time schedule of s/w releases.

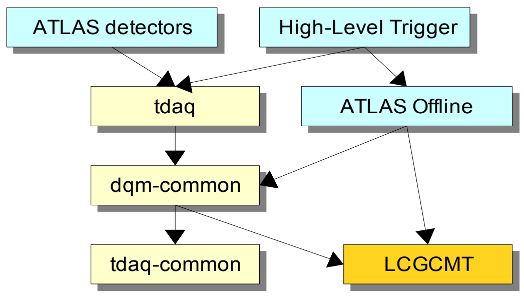


Figure 2: integration of TDAQ s/w projects with other ATLAS projects and external s/w.

2.3.2 External and Third-party s/w

The following third-party s/w packages are widely used within TDAQ s/w:

Boost (version 1.39), Python 2.5, Qt4, Java 1.6.0, Oracle v10 client, ROOT 5.0.22.

These packages are built and maintained by other teams and are accessed by other projects from LCGCMT project (LCG, 2010).

Some other more specific packages are integrated and built within TDAQ release: OmniORB (a CORBA implementation), CLIPS (expert system framework) and maintained as other TDAQ s/w packages.

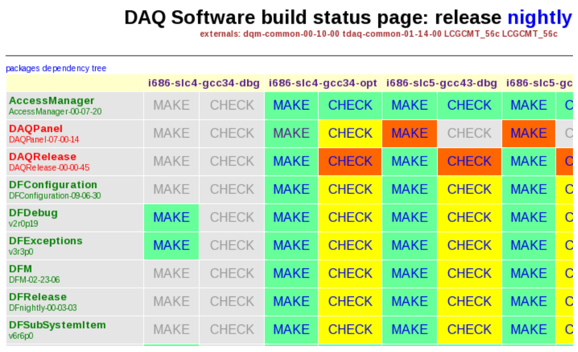
2.4 Nightly Builds

Regular "nightly" builds are performed on a shared filesystem during the night hours, so every day developers access the most fresh versions of all packages built together as "nightly" release.

Nightly builds are the main area for the integration of new developments and the target for regular automatic validation tests.

Figure 3 shows a fragment of the web page which displays the results of a nightly build in form of a table. In the first column all packages and their versions used in the build are listed. Other columns correspond to the platform used in this build, e.g. i686-slc4-gcc-34-opt. For every package and platform combination, the log files from the make and check targets are accessible.
In case of failure of the make or of the check target of a package, an e-mail is sent to the responsible developers of that package, as defined in the package configuration file.

When a particular nightly release is successfully built for all essential packages and platforms and passed some validation checks, it is installed in more permanent area and thus it can used by developers in case the most fresh nightly build is failed by some reason.



DAQ Software build status page: release nightly

externals: dqm-common-00-10-00 tdaq-common-01-14-00 LCGCMT_56c LCGCMT_56c

packages dependency tree

	i686-slc4-gcc34-dbg		i686-slc4-gcc34-opt		i686-slc5-gcc43-dbg		i686-slc5-gc	
AccessManager AccessManager-00-07-20	MAKE	CHECK	MAKE	CHECK	MAKE	CHECK	MAKE	C
DAQPanel DAQPanel-07-00-14	MAKE	CHECK	MAKE	CHECK	MAKE	CHECK	MAKE	C
DAQRelease DAQRelease-00-00-45	MAKE	CHECK	MAKE	CHECK	MAKE	CHECK	MAKE	C
DFConfiguration DFConfiguration-09-06-30	MAKE	CHECK	MAKE	CHECK	MAKE	CHECK	MAKE	C
DFDebug v2r0p19	MAKE	CHECK	MAKE	CHECK	MAKE	CHECK	MAKE	C
DFExceptions v3r3p0	MAKE	CHECK	MAKE	CHECK	MAKE	CHECK	MAKE	C
DFM DFM-02-23-06	MAKE	CHECK	MAKE	CHECK	MAKE	CHECK	MAKE	C
DFRelease DFnightly-00-03-03	MAKE	CHECK	MAKE	CHECK	MAKE	CHECK	MAKE	C
DFSubSystemItem v6r6p0	MAKE	CHECK	MAKE	CHECK	MAKE	CHECK	MAKE	C

Figure 3: Web page with nightly release build table (fragment).

2.5 Developers Guidelines

A typical working mode of a developer includes the following steps:

- check-out a package (or few packages) from code repository
- prepare a package and build it against a release
- check the functionality of the package
- commit and tag the package in the code repository
- submit the tag for the release build

Normally developer has a number of packages in his working area. The mechanism is provided to build the run-time environment such that this area is taken into account when developer runs the s/w in order to validate the changes before committing the changes back to the code repository.

3 DEVELOPMENT TOOLS

3.1 Code Repository

The SVN code management system (SVN, 2010) and repository is used for keeping the user's code and tracking the changes. In the earlier stages, CVS system was used, but since the global CERN policy and support was moved to SVN, TDAQ project moved the code repository to SVN as well. The web-access to the repository is made available via Trac [http://trac.edgewall.org/] and WebSVN systems [http://svnweb.cern.ch/world/wsvn].

3.2 Development Tools

3.2.1 CMT: Configuration Management

CMT stands for Configuration Management Tool (CMT, 2010). It was approved as a principal tool for configuring the build and the runtime environment in ATLAS.

CMT is not a full-featured s/w release tool, but rather a low level configuration tool. A number of policy files, make fragments and scripts were developed in order to implement all the functionality required for a release build tool.

CMT allows to describe what is provided by a package in a special 'requirements' file and to avoid writing ordinary makefiles for different platforms. A requirements file includes all settings for building libraries, applications and custom targets, and also for defining the runtime (shell) environment which is used for running the s/w. Essentially CMT provides a number of *make* fragments, such that they are built together and made used from the top-level makefile when developer runs the standard "make" command.

CMT packages can "use" other packages, thus giving the possibility to create dependencies between packages and to organize a group of packages - in other words called "releases". The build and run-time settings are inherited through the "use" chain.

3.2.2 Compilers and Build Platforms

The GNU compiler collection (GCC) is used as a primary compiler for the s/w. The current version in use is 4.3.2 on SLC5 Linux. Given the fact that TDAQ s/w is used in scope of other projects and also uses external s/w (see Section 2.3.2), the supported version of the compiler is defined at the experiment level.

The build platform tag is composed of 2 parts: the h/w part which defines the hardware and the OS (e.g. i686-slc5) and the s/w part which defines compiler and compiler options (e.g. gcc43-opt).

Later these tags are used to select which binaries can be started on particular nodes in the TDAQ system.

The following table summarizes currently supported build configurations.

Table 2: Supported build platforms.

Tag	Details
i686-slc4-gcc34-opt	32 bit, SLC4 Linux, gcc34 compiler, optimised code
i686-slc4-gcc34-dbg	debug (-O0 -g)
i686-slc5-gcc43-opt	32 bit, SLC5 Linux, gcc43 compiler, optimised code
i686-slc5-gcc43-dbg	debug
x86_64-slc5-gcc43-opt	64 bit, SLC5 Linux, gcc43 compiler, optimised code

3.2.3 Distributed Compilation

To achieve maximum performance of the releases builds, a number of techniques is used:

- release for each platform is build independently
- independent packages within a release are built in parallel, using CMT *tbroadcast* tool
- a combination of multi-job feature of gmake ("-jN") and *distcc* (distributed CC) is used to fully utilize a cluster of multi-core build nodes

The described approach allows to build the whole TDAQ release for one platform in 1-2 hours.

3.2.4 Installation Policy

After the build, all items which a package is contributing to the release are installed in a common installation area, according to a pre-defined layout. Typically the sequence of gmake commands issued by a user looks like

```
> gmake && gmake inst
```

A number of installation pattern are defined in the TDAQ CMT policy package, allowing developers to install different types of files according to a predefined layout of the installation area. The following type of files may be installed: binaries (libraries, applications); scripts; Jar archives; Python scripts and libraries; package data files (e.g. images); examples; documentation.

The common installation area essentially forms the distribution of the release which can be exported for external use.

3.2.5 Check Target

Every package may provide a way to test it's basic functionality e.g. by building and executing some test application or by launching more complex scenario from a script. This is done in special "check" target which is normally launched after the installation phase.

```
> gmake && gmake inst
> gmake check
```

During the nightly builds, check targets are launched automatically for all packages, thus helping to spot some basic problems with the s/w.

3.3 Documentation

Standard LXR and Doxygen documentation (for Java and C++) are generated for every release, including nightly builds.

In addition, each package provides "release notes" which are distributed with the package, and also a combined digest of all changes in the release is generated.

3.4 Debugging and Profiling Tools

The memory debugging and performance profiling tools are important to guarantee the quality of the s/w. We widely use *valgrind* (Valgrind, 2010) tool for finding memory allocation problems and for code performance profiling ("*cachegrind*"). Unfortunately this tool is difficult to use to debug race-condition related errors in a network and multi-threaded environment environment, because it executes the code in scope of a virtual machine which slows it down by few tens times.

4 DISTRIBUTION AND PATCHING

4.1 RPM: Distribution and Installation Tool

Initially the release was distributed as a number of "tar-ball" files by a helper script. Later, we have chosen the standard in the Red Hat Linux world RPM (RedHat Package Manager, (RPM, 2010)) as the main distribution and installation tool.

During the installation phase of the make process, a record is kept on which files get installed in the installation area and later this list is given to RPM build tool to build RPM packages. Every CMT TDAQ package is packaged as a number of RPM packages, according to the following convention:

```
<release>_<package>_<tag>-<version>.noarch.rpm
```

Where <tag> is either the binary tag for binaries for particular platform (tag), or “noarch” for platform independent file or “src” for package sources. Such scheme gives the flexibility to install only required subset of the s/w release.

Given the number of involved projects, packages and dependencies between them, to facilitate the installation procedures, the *apt* repository manager is used (Apt, 2010). It is capable to access a number of RPM repositories, resolve dependencies and download packages for further local installation with RPM.

4.2 Patching Policy and Experience

Initially the whole release was distributed as a small number of big RPM packages without the possibility to install or update an individual package. This caused problems for the patching policy for TDAQ s/w, where it was required to have a possibility to easily install a new ("patched") version of a particular package without disturbing the rest of the s/w and also a possibility to roll-back the patch i.e. to install the previous version of the package.

To fulfil this requirement, the granularity of packaging was changed such that each CMT package can be distributed independently. CMT version of a package (which corresponds to tag in SVN) is transformed in the RPM package version. With such schema, a *patch* for the release is just a new RPM version of a particular package. Thus, a package can be upgraded or downgraded in a relatively short time, which is essential in the condition when the intervention to the running system is very limited in time and must be done smoothly and as quickly as possible.

The building of new versions of packages are fully automated, developers simply needs to submit a new tag to the build system.

Currently the joint ATLAS s/w RPM installation at ATLAS pit contains about 2500 packages holding 2.5 million files which take almost 90 Gb of the disk space. Such a scale makes it a challenge for the packaging tool.

5 CONCLUSIONS

The paper presented an overview of the model, policies and tools for software releases management in a big ATLAS TDAQ project. The feature of this project is that it's s/w has being actively developed through last 10 years by a big distributed team of developers and at the same time was widely used in validation and production environment of the ATLAS experiment, where it will be maintained in another 10 years.

It is shown that the developed s/w release model and tools fulfil this demanding requirements and conditions. The key aspects of the model and the basic tools used for the implementation are described, emphasising the policies used for maintaining the s/w on a productions system through the coming years of running the ATLAS experiment.

REFERENCES

ATLAS Collaboration, 2003. ATLAS High-Level Trigger Data Acquisition and Controls Technical Design Report, *CERN/LHCC/2003-022.*

Gadomski, S. et al, 2006. Deployment and Use of the ATLAS DAQ in the Combined Test Beam*, Nuclear Science, IEEE Transactions on Volume: 53 , Issue: 4 , 2006 , Page(s): 2156 – 2161.*

LCG, 2010. LCG-AA Software Elements, Available at http://lcgsoft.cern.ch/ [Accessed 30 March, 2010]

SVN, 2010. Central SVN Service, Available at http:// svn.web.cern.ch/svn/index.php [Accessed 30 March 2010].

CMT, 2010. CMT Configuration Management Tool. Available at http://www.cmtsite.org/ [Accessed 30 March 2010].

Valgrind, 2010. Valgrind 3.3 -Advanced Debugging and Profiling for GNU/Linux applications. Available at http://www.network-theory.co.uk/valgrind/manual/ [Accessed 30 March 2010]

RPM, 2010. Maximum RPM, Available at http:// www.rpm.org/max-rpm/ [Accessed 30 March 2010].

APT, 2010. Advanced Packaging Tool, Available at http:// en.wikipedia.org/wiki/Advanced_Packaging_Tool. [Accessed 30 March 2010].