ORGANIZATION AND MANAGEMENT OF ATLAS SOFTWARE RELEASES

Solveig Albrand, Johann Collot, Jerome Fulachier, Fabian Lambert, LPSC, IN2P3/CNRS, Grenoble, France. Christian Arnault, Vincent Garonne, Arthur Schaffer, LAL/IN2P3/CNRS, Paris, France. Eric Nzuobontane, Peter Sherwood, Brinick Simmons, University College, London, UK. Simon George, Grigori Rybkine, Royal Holloway, University of London, UK. Steve Lloyd, Queen Mary, University of London, UK. Emil Obreshkov, Institute for Nuclear Research and Nuclear Energy (INRNE) Bulgarian Academy of Sciences, Bulgaria. Alessandro De Salvo, INFN - Roma1, Italy. Hans von der Schmitt, Max-Planck-Institut für Physik (Werner-Heisenberg-Institut), München, Germany. Vasily Kabachenko, Institute for High Energy Physics (IHEP), Protvino, Russia. Zhongliang Ren, Di Qing, Institute of Physics Academia Sinica, Taiwan. David Quarrie, LBNL, CA, USA. Traudl Hansl-Kozanecka, University of California, Santa Cruz, USA. Frederick Luehring, University of Indiana, USA. Alexander Undrus, BNL, Upton, NY, USA. Edward Moyse, University of Massachusetts, Amherst, MA, USA. Saul Youssef, Boston University, MA, USA. Steven Goldfarb, University of Michigan, USA.

Abstract

ATLAS is one of the largest collaborations ever undertaken in the physical sciences. This paper explains how the software infrastructure is organized to manage collaborative code development by around 300 developers with varying degrees of expertise, situated in 30 different countries. We will describe how the succeeding releases of the software are built, validated and subsequently deployed to remote sites. Several software management tools have been used, the majority of which are not ATLAS specific; we will show how they have been integrated.

ATLAS offline software currently consists of about 2 MSLOC contained in 6800 C++ classes, organized in almost 1000 packages.

INTRODUCTION

ATLAS is one of the largest collaborative efforts ever undertaken in the physical sciences. About 2000 physicists participate, from more than 150 universities and laboratories in more than 30 countries.

The software is correspondingly large both in size, and also by the number of developers involved. These considerations have lead us to put into place policies and technical methods to make the development and release mechanism as smooth and efficient as possible.

A release of ATLAS offline software contains approximately 1000 code directories containing about 2 MSLOC. The code is almost exclusively in C++ although some legacy FORTRAN code remains. Job options files for run time control are written in Python.

Currently about 300 members of the collaboration are registered as involved in development, although of course many of them are not engaged in this activity full-time. However the number is expected to increase as we move towards real data taking, as effort previously devoted to building the detector will become available.

On the other hand, we expect the total size of the release to be reduced, as this is normal in the development process as software reaches maturity.

Since the first release of ATLAS off line software for the Physics Technical Design Report in May 2000, 12 major releases of the software have been produced.

By 2001, it became clear that the difficulties involved in managing software produced by a large number of people who are widely distributed geographically and in their majority not expert software engineers, required the establishment of a management structure, and a coordinated set of tools. The ATLAS software infrastructure team was put into place to provide the technical support necessary. This paper explains the organization we have put in place, the different tools we use, and how they work together.

RELEASE ORGANIZATION

These parties are involved in the release in either a management or a technical capacity:

- The Chief Architect directs the software effort and takes the final decisions on the software design and the schedule for the introduction of new features of the software or the release structure.
- The Software Project Management Board (SPMB) approves major policy decisions in consensus with the Chief Architect and serves as a forum for discussion of policy. All interested communities, such as the sub-detector groups, the testbeam coordination, the physics working groups and the software sub-domains have representatives on the SPMB.
- The Software Infrastructure Team (SIT) is the technical group which brings together the

providers and the users of the tools used to build the software and its distribution kit. The group meets every two weeks to discuss coordination issues and schedules.

- The Software Librarians are the principle users of the build tools; they actually build the software releases and the kits, and they oversee the distribution.
- The Release Coordinators have a policing role. They must validate that software submitted by each community functions properly and meets the design goals of each release. They can ask for changes in the submitted code or refuse to accept code. They decide when a release is ready to be built. Release coordination is a heavy responsibility, so we try to rotate this role so that no one person holds it for longer than the time it takes to build a major release.
- The Package Managers and Package Developers manage and write the software that we produce. For smaller packages these roles may be held by the same people. Larger packages have a hierarchical organization whereby new code must be approved by a manger before inclusion in the release.

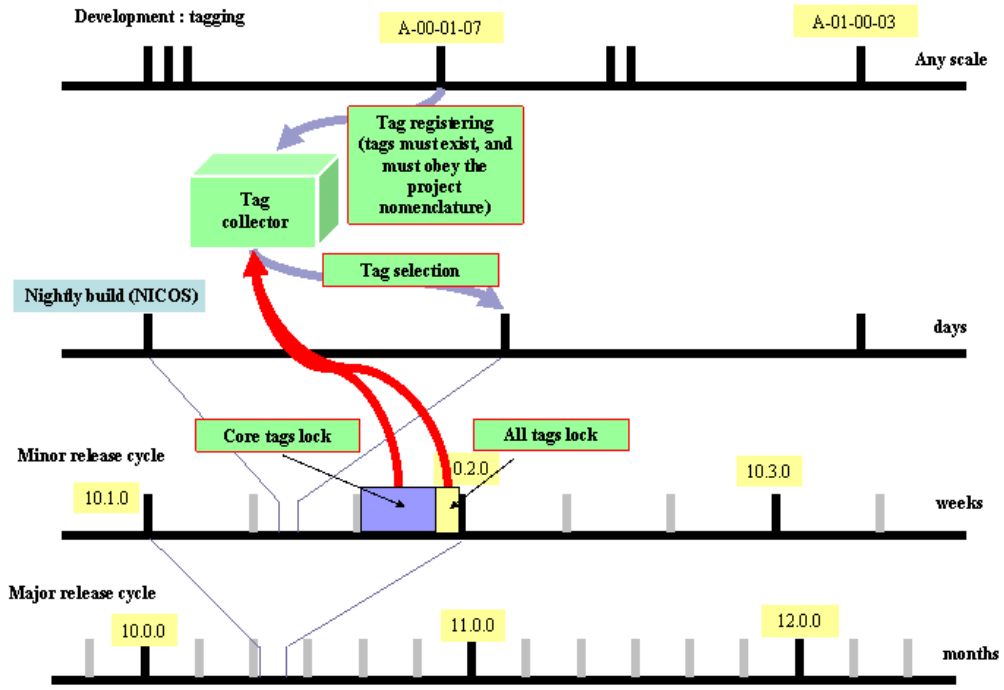

Figure 1 : The software life cycle - shows the different time scales for different types of releases.

RELEASE POLICIES AND THE RELEASE CYCLE

ATLAS uses several kinds of builds in order to make a release. As we approach a production release it becomes more and more difficult to insert new packages, or new versions of packages in the software. Indeed, we have a mechanism to lock parts of the release which prevents such an insertion. Figure 1 shows the chronology of these different releases, and introduces two of the tools we use; Tag Collector [1], which is a database application which holds the list of the package versions which make up the release, and contains some locking functions, and NICOS [2], which runs the nightly builds.

Every 24 hours there is a nightly build of the software. Usually the "nightly" is an incremental build (to save time) but once a week, it is a full build of all packages (~20 hours). Developers can generally put code freely into a nightly release, and it is not essential for the configuration file to be correct. The main purpose of these builds is to test the integration of the different packages.

Every 3-4 weeks there is a development or minor release of the software. New code is expected to follow the software design but developers do have considerable freedom in what code they submit. These builds provide a stable version to develop against, but are not rigorous production quality releases – they are not validated for use in physics.

Every 6 months there is a production candidate or major release which provides the basis of the software used in the next production. Restrictions are put on what code can be added to the production release.

In general we are obliged to follow each production candidate release by one or more bug fix releases that fix

problems found during testing and validation of the production candidate release and the succeeding bug fix releases. Very strict rules for adding code are followed.

THE DEVELOPEMENT PROCESS

We describe here the cycle of development and tool usage which are used by ATLAS to manage software.

Submitting a new version of a package

Using CMT [3], a configuration management tool, a developer checks out a package from the CVS repository and works on it, building against one of the releases which already exist. When he is satisfied, the code is committed to CVS, and tagged. Next, if the new package version is to be included in a release, the tag must be entered into the Tag Collector database. The user logs on to the Tag Collector, and is presented with a list of releases which are currently open for tag collection. If the build of a production release is pending, tag collection may be locked, and it is impossible for the developer to make any addition; he must request that the operation be done by the release coordinator.

Building and distributing the release

The nightly build system, NICOS, and the librarian, who uses pure CMT commands to make a release, both obtain the list of tags of package versions to be included in the release from the Tag Collector. The essential difference between a nightly release build and a numbered release build is that the Container packages (e.g. Event, Database) are not used to determine the set of packages and tags for the nightly releases, but the set is taken directly from the leaf packages which contain the source code. In contrast, developer or production releases use the information from container packages and consistency is enforced between the set of packages and tags derived directly from the Tag Collector, and that derived hierarchically by CMT from the Container packages.

Once a release suitable for physics has been produced, the librarian prepares the distribution kit using a suite of shell and python scripts developed for this purpose [4]. The kit is placed on a disk cache accessible via http. Grid site managers or individuals can use Pacman [5] to fetch and install ATLAS software. The kit includes a validation suite.

Automatic Documentation

Doxygen documentation is produced for every release, and we have made considerable effort over the past year to improve its quality. A large number of packages now include a “mainpage” which provides an overview of the package and its interfaces. Other documentation initiatives are described below.

Quality Assurance and Testing

Unit and integration tests are managed by NICOS. Regression tests are run by the Run Time Tester (RTT) [6]. Tests are run after the completion of all releases. We also run some other QA tools over the release. They are described in more detail below.

SOME DETAILS OF TOOLS USED BY ATLAS FOR RELEASE MANAGEMENT

Configuration Management

The tool ATLAS uses for configuration management is CMT [3]. This tool supports the decomposition of the software base into packages, or groups of packages. Every package or package group must specify its configuration in a textual requirements file. The configuration defines the dependencies on other packages (“use” statements) and the parameters needed to run tools like “make”, Doxygen or kit building scripts. It also describes how to make the release components such as applications or libraries, how to use them, for example how to find the application job options, and how to apply management actions (build, documentation generation, tests, installation etc.). Management actions are broadcast through the software hierarchically following the dependency tree.

Tag Collection

The ATLAS Tag Collector [1] is a database application with a powerful web interface. A web service is also provided. It provides management tools to enforce policy, such as the ability to lock parts of the release, and tools for release building. In particular both CMT and NICOS use the web service interface to obtain the list of package versions which must be built together. Tag Collector is also able to construct the correct CMT requirements file for a package group, and to tag package groups automatically in the code repository.

Tag Collector was designed for ATLAS, but in a generic way; the links to CVS and to CMT are decoupled from the main code.

Tag Collector is part of the ATLAS metadata framework, and as such provides required information on releases and code packages.

Nightly Builds

The NICOS tool runs both a debug and an optimized build for each open release every 24 hours. Each build has a 7 day lifetime. The results are returned to NICOS for error analysis, and any problems detected are relayed by e-mail to the registered developer. NICOS provides a set of detailed web pages showing the status of all releases and also the results of the unit and integration tests run after the build.

The tool contains automatic discovery and recovery of build failures. It builds on reliable local disks before copying to the developers’ release area.

Kit Building and Distribution

The packaging unit chosen for kit building follows the CMT package granularity. The kit compressed tar ball includes binaries, header, configuration, and data files.

This meets both production and development use cases. The Pacman [5]file references the packed software and describes how it should be fetched, installed, configured and updated. Pacman aims to maintain the integrity, repeatability and verifiability of installed releases at remote sites.

Testing

As mentioned above, NICOS includes a mechanism for running both unit and integration tests. Regression tests are managed by the Run Time Tester (RTT) [6]. This powerful tool runs tests, which are defined by developers, after each build is created. Jobs are constructed, and then run under surveillance on either a batch farm, or on a Grid element. Results are captured, and analysed, and reported to the user via a detailed set of web pages. It is possible to download the job result files or log files for a detailed analysis.

Our testing tools have agreed on a common test description language. Tests are defined within the CMT framework, and are read by NICOS, RTT and the Kit validation mechanism

Quality Assurance Tools

A tool named "checkreq" is run over all packages to verify the consistency of the declared package dependencies, and the #include instructions of the C++ code.

RuleChecker, a tool to check compliance with many of the ATLAS C++ coding rules can be run on all or parts of the release source code using RTT. At present this is done on a volunteer basis.

For further information on ATLAS quality assurance please see reference [7].

Documentation

The past year has seen an enormous effort to improve the quality of ATLAS documentation. Building Doxygen documentation is done for every release of ATLAS software.

There has been a total remodelling of the ATLAS computing pages, with effort to bring up to date the contents, and to improve and unify the format [8]. The pages follow latest recommended web standards, separating content and style by using style sheets.

Another very popular initiative has been the ATLAS workbook [9]. This is a set of WIKI pages updated for each production release, which is a beginner's guide to the software. It provides information on everything from how to obtain an ATLAS account, to how to run physics jobs on the grid. A PDF version of the workbook can be generated.

RECENT IMPROVEMENTS AND FUTURE PLANS

The ATLAS software release is large. A monolithic build currently takes ~20 hrs. To alleviate matters we have divided the software into "projects". A project is a group of packages which can have its own independent development timeline. Higher level projects can develop against stable versions of lower level ones (closer to the core software). The division of the release into projects was introduced in January 2006; but it is not yet finished. It has implied modification of all tools.

A new kit building scripts suite was developed to construct the distribution kit for each project. Packaging with the granularity of CMT projects, the suite has reduced kit-building time significantly. The project kit is sub-divided into platform-dependent (optimized, debug), platform-independent, source and documentation parts. This permits a switch to using the distribution kit for CERN installations as well as remote sites.

We expect to see an increasing emphasis on code testing and validation by increasing the use of the RTT, extending its use to grid nodes.

Coding Rules will be verified by a more strict application of the RuleChecker tool.

Finally, following a detailed evaluation of the "Subversion" repository management tool, we will decide over the next few months if what we might gain by replacing CVS is greater than the likely perturbation of release production.

CONCLUSION

ATLAS has developed a suite of tools for code release that works well with our base of about 300 code developers spread out over the world.

The suite, combined with our management structure, allows us the flexibility to respond to the inevitable crises in building releases without losing control of the release process.

Most of this suite has been developed and integrated since the advent of the Software Infrastructure Team (SIT).

The SIT has taken the critical central role in integrating the tools and using them to make over 50 Atlas offline software releases. The value of having this dedicated team leading the software releases can not be overstated.

REFERENCES

[1] https://atlastagcollector.in2p3.fr
[2] http://www.usatlas.bnl.gov/computing/software/nicos/
[3] http://www.cmtsite.org/
[4] http://atlas-sw.cern.ch/cgi-bin/viewcvs-atlas.cgi/offline/Deployment/
[5] http://physics.bu.edu/pacman/
[6] http://www.hep.ucl.ac.uk/atlas/AtlasTesting/
[7] http://cern.ch/atlas-computing/projects/qa/qa.php
[8] http://cern.ch/atlas-computing/computing.php
[9] https://twiki.cern.ch/twiki/bin/view/Atlas/WorkBook