Automated software packaging and installation for the ATLAS experiment

Simon George[1,*], Christian Arnault[2], Michael Gardner[1], Roger Jones[3], Saul Youssef[4]

[1] Department of Physics, Royal Holloway, University of London, TW20 0EX, UK; supported by PPARC.
[2] LAL Orsay
[3] Department of Physics, Lancaster University, Lancaster LA1 4YB, UK.
[4] Boston University
[*] Speaker

Abstract

Managing the distribution and installation of large and complex software suites such as that of the ATLAS Particle Physics experiment [1] gives rise to a variety of problems. To be able to deploy software on the Grid it must have a completely automated installation procedure that addresses issues such compatibility, updates and external software required. Installations are needed to satisfy the different requirements of the production run, a developer (with his/her own code) and the possibility to rebuild from source.

A solution to this problem has been developed in the context of ATLAS software. ATLAS uses CMT [8] to configure and manage packages, which are the basic units of the software. Crucially for this project, CMT allows the definition and imposition of conventions for package metadata. Pacman [9] is a package manager that facilitates transparent fetching, installation and management of software packages. These two pieces of widely used software provide much of the functionality required to meet the packaging, distribution and installation needs of ATLAS; the main work is to interface them and solve some problems with the ATLAS software itself. To this end, additional tools have been developed to extract files in common package formats (such as tar and RPM) and write the Pacman files. These files describe all the information needed to fetch and install a package, including its dependencies on other packages. External packages are handled by additional metadata written in interface packages, thus allowing them to be packaged in a similar way. The presence of basic system packages is tested before the download and installation commences.

1. Introduction

ATLAS [1][2] is a general-purpose particle physics experiment that will study topics including the origin of mass, the processes that allowed an excess of matter over antimatter in the universe, evidence for Supersymmetry and other new physics. The experiment is being constructed by some 2000 scientists in over 100 institutes in 6 continents. The experiment will be located at the 27 km circumference Large Hadron Collider at CERN, the European Laboratory for Particle Physics near Geneva.

A large software suite has been developed to simulate and analyse data from the experiment. It consists of around 2.5 GB of files organised into about 500 *packages*, including around 50 *external* packages which are not developed within the project. Production releases are made approximately every 3-4 months, intermediate developer releases every two weeks and snap-shot builds every night.

It takes about 10 hours to build a release, but only about 1-2 hours to download and install a pre-built kit. This depends of course on the speed of the network and local disk access.

Production releases will be used for large-scale simulation, data mining and data analysis. Some analysis is also done with intermediate releases. Developers need access to the latest releases and sometimes the latest night's build. The primary installation site is CERN, and a few other sites maintain mirrors of this by semi-automated processes.

Two 'Data Challenge' exercises have recently been carried out [3][4] to gain experience with the Grid working model, test software tools, and harness available computing resources of ATLAS collaborators around the world for the production of large quantities of simulated data. This required deployment of the ATLAS

software at participating sites. A first generation of installation kits were made for this purpose, using several different methods: see for example [6]. The experience of this has been fed into the development of new procedures and tools. This paper describes the new approach taken to packaging and installation and how the second generation kits will be produced.

2. Requirements and constraints

In the language of GLUE [7], we are talking about *packaging*, *distribution* and *configuration* tools. Before packaging, it is assumed that the software has been built with *building* tools.

2.1. Types of kit required

The need for several different types of installation kit has been identified:

- a pre-built binary kit for data challenges and productions on a supported platform;
- a pre-built developers' kit which software can be built and run against;
- a full source kit will be needed to install the software on non-standard platforms, and for sites that want to be able to browse the full source.

Just like releases, kits will have to be made for all supported ATLAS platforms, compilers and configurations (e.g. optimised and debug builds). Currently[1], the main platform for ATLAS is Intel x86/Linux (RedHat 7.3). Sun SPARC/Solaris 8 is also supported, but not Intel/Windows.

Some developers would like the possibility to install everything needed to run a release on a laptop, such that it can subsequently be used without a network connection.

2.2. Installation

Site managers who need to install the software want push-button deployment and automatic configuration, which can be run unattended. They need the freedom to locate the kit anywhere. Scripts to set up the local user environment should be included. Due to the large size of the ATLAS software, duplicate installation of unchanged packages is to be avoided. Site managers would like clear list of prerequisite software that is not included in the distribution and preferably some automatic check for this. This list should not be long nor contain anything too obscure. Apart from this, once installed, kits should be self-contained, so for example they do not need to access files on a network file system such as AFS or remote databases.

Users need to be able to install the ATLAS software in their own file space without any special access permission (e.g. "root" access).

The actions of the tools should be reversible, i.e. tools should be able to unconfigure and remove software.

2.3. User environment

Once installed and configured, there must be a simple means to set up the user's environment. Typically, his involves setting standard environment variables in the user's shell so that executable programs and libraries may be found (the environment variables `PATH`, and `LD_LIBRARY_PATH` respectively) along with any other environment variables or configuration files that are specific to the software installed. User environment is an important issue that configuration management attempts to solve.

2.4. The software itself

The ATLAS software contains two distinct types of package: *internal* and *external*. Internal packages are developed and managed within the ATLAS software project. External packages are software obtained from the Particle Physics community or from the public domain, or can be commercial products.

A *release* comprises a set of internal software packages along with references to any external software that is needed, but not the external software itself. External software is built and installed as needed, asynchronous to the release cycle.

The existing release and distribution procedure is as follows. A release is first built and installed at the primary site using CMT. Once testing is successfully completed, the same release is built from scratch and installed at a few other sites, following the same procedure, but this often requires manual intervention. The reason is usually an assumption about the location of external software which is only valid at the primary site, and the lack of a standard distribution mechanism for external software. External software packages are by their very

[1] The list of supported platforms and level of support is occasionally reviewed. Influences include what equipment is available and market trends.

nature inconsistent in layout, installation and use. One of the challenges of software distribution is to find a way to handle this uniformly.

2.5. Tools in use

ATLAS already uses the Configuration Management Tool (CMT) [8] to manage configuration and building of its software and set up the user environment. This is described in the following sections.

3. Choice of tools

3.1. CMT

Given that ATLAS was already using CMT [8] for configuration management and building releases, it was an obvious choice to exploit this. However, it is worth explaining why CMT is particularly useful for this job.

CMT is a configuration management environment and comprises several shell-based utilities. It is an attempt to formalize software production, especially configuration management, around a package-oriented principle. The notion of packages represents a set of software components (e.g. applications, libraries, tools) that are to be used for producing a system or a framework. In such an environment, several persons are assumed to participate in the development and the components themselves are either independent or related to each other. The environment provides conventions (for naming packages, files, directories and for addressing them) and tools for automating as much as possible the implementation of these conventions. It permits description of the configuration requirements and automatically deduces from the description the effective set of configuration parameters needed to operate the packages (typically for building them or using them). It also derives and provides a tool to set up the user environment necessary to do this. CMT determines dependency trees and can 'visit' every package in a tree to apply a command.

CMT has a broad user base, especially in Particle Physics and Astronomy software projects.

3.2. Pacman

Pacman is a package manager. It can be used to define how the software they wish to distribute should be fetched, installed, configured and updated. The packages can be any mix of tar-balls, RPM or other types. Pacman doesn't alter these; instead the description goes in a separate 'Pacman file' that references the package itself. A directory of Pacman files, usually accessible via the web, is known as a 'Pacman cache'.

So the cache contains instructions on how to fetch and install software, but not the software itself. This can be anywhere else on the web. Caches themselves are distributed and cross cache dependencies are possible.

Pacman provides a flexible way to require or test for the presence of prerequisite software. If the test fails, it will stop *before* installing the package and explain the problem. After installation, Pacman provides a web interface showing the status of all the installed software, which would be convenient for remote monitoring.

The advantage of using Pacman is that all the details of installing software are taken care of by the packager of the software, not the person who installs it, thus reducing the total effort to deploy software. The philosophy of Pacman is to put the onus on the packager to make a fool-proof cache file and solve problems that users have.

Pacman is already used by several Particle Physics experiments and GRID projects.

3.3. Package format

The first installation kits were binary only, for Intel x86/Linux, and used the RPM [10] format. RPM is a nice solution for packaging and installation that is widely used with several popular Linux distributions, although there are some inconveniences associated with it. With the default configuration it will only allow software to be installed by "root", although there is a way to work around this. RPM is not commonly used on other operating systems.

With the functionality already provided by CMT and Pacman, simple "tar" archives [11] are sufficient for packaging files and can be expected to work across a wide range of machines. The tar files are compressed with 'gzip' to save disk space and reduce download time.

The functionalities of CMT, Pacman and tar are conveniently complementary and they do seem able to meet all the requirements. On the other hand, RPM has some overlap with each of them so it was retained as a second option for the package format. The remaining task is to make these tools work together to provide the installation kit.

4. How it works

For each package, a shell script is run to obtain the information needed and create a Pacman file, tar file and, optionally, RPM file. This process will be explained in detail, after which the general procedure to create, install and use a kit is described.

4.1. Packaging granularity

As ATLAS already uses CMT, the software is already organised into a hierarchy of inter-dependent packages. One question was whether this was the best granularity for packaging and installation. For production purposes, it would be just as good to install the entire ATLAS software suite as a single package, and perhaps simpler. However, this makes it harder to split the software into several sub-releases, which is planned. Furthermore, the way Pacman works, only one package has to be requested, and then all those packages it depends on are installed automatically, so the number of packages has no impact on the ease of installation. Finally, it would be convenient to maintain similarity between source kits and pre-built binary kits. It was therefore decided to retain the full granularity.

4.2. Meta-data provided by the packages to construct the distribution kits

The meta-data needed to construct the distribution kit of a given package are expressed inside its CMT requirements files (a sample is shown in Figure 1). They are either directly deduced from implicit CMT specifications (such as dependencies, or constituent definitions) or from conventional parameters (i.e. CMT macros) parameterised by the current configuration (platform, site, etc.).

- The dependencies, specified by CMT "`use`" statements, are used to generate dependencies either in the RPM or Pacman manifest.

Internal packages generally follow the ATLAS policy for location of their binaries, of their header files, etc and they describe their constituents (applications or libraries) in their requirements files. The meta-data they provide are:

- Applications and libraries (i.e. constituents) using the corresponding CMT statements (application and library).
- Run time files, i.e. conventionally all files from `../share`. It is expected in the next developments that other locations can also be specified, using a dedicated CMT macro.
- XML files specified in the `use_xmls` CMT macro.
- Header files, when the package provides them in the conventional location, `../<package>`

External packages are actually pure meta-data packages: they only provide descriptions that allow some external software to be handled in the same way as internal packages. The real software or the package can be at any arbitrary location that is accessible when building or packaging is done. The meta-data they provide are therefore the list of file paths or directory paths that should be exported into the distribution kit of the package. This is specified in one conventional CMT macro named `cmt_export_paths`. This macro should be declared in such a way that the value it takes at installation time should be relative to one single environment variable named `SITEROOT`.

4.3. Directory structure of installed files

The structure of the kit itself is designed to be relocatable, and able to support the incremental installation of all packages below one single installation area.

The entire software base is structured in terms of sub-projects (following the usual software base structuring expressed in the `CMTPATH` environment variable). Each project is assigned one global version identifier. Currently ATLAS has two sub-projects:

1. the release of the internal ATLAS software;
2. Gaudi [12] (the generic Particle Physics software framework).

Any package may either belong to one of the sub-projects or be an external package.

The resulting internal directory structure of the kit reflects this organization in terms of sub-projects and external packages as follows:

1) The CMT specification of each package in a sub-project is stored so as to replicate the original directory structure for all CMT metadata files. This set of metadata files form the complete configuration set for the software, and remains separated from the package contents.

2) The contents of each package in a sub-project are stored in one single installation area below the sub-project directory

3) The contents of each external package are stored in one top directory per package

One sees that:

- this structure is designed to be identical to the normal installation area of the ATLAS software base;
- both internal and external packages may be incrementally installed below one unique and shared root location (often designated as `SITEROOT`);
- the installation is independent of this root location (it is relocatable);
- several versions of the sub-projects may be installed below the same root location;
- the binary files corresponding to several different platforms may be incrementally installed within the same structure.

Installing the kits amounts to expanding them below one given root location, so to produce a complete working system it is therefore a priori sufficient:

- to set an environment variable `SITEROOT` pointing at this root location;
- to fill in the `CMTPATH` variable with the proper set of <project>/<project-id> entries in the software base.

Then, CMT will automatically find all the information it needs to set up the user's environment (such as `LD_LIBRARY_PATH`, `PATH`, other environment variables, include search path, configuration file search path, etc…), so the development kit can be used in the same way as the original installation from which it was built.

4.4. Pacman files

A typical Pacman file, for a package called 'AthExHelloWorld', is shown in Figure 2. It was generated automatically using the tools described, from the CMT requirements file shown in Figure 1. Note that the source path, shown in the example as a relative path to a directory called `dist`, is one of several arbitrary parameters of the script that creates the kit.

4.5. Creating the installation kit

The kit should be created by the central librarian as the part of the release procedure. Everything described below can in fact be scripted. The first step is to select a release and specify its location.

An ATLAS release is defined by an empty package called "AtlasRelease" that just has dependencies on the set of packages and versions that form the release. CMT is used to visit every package in the release (the '`cmt broadcast`' command) and there run the script to create a Pacman file, tar file and, optionally, RPM file for the package.

An option when walking the dependency tree is to walk across different project areas (different directories in `CMTPATH`). This option allows the ATLAS distribution to include kits for other projects such as Gaudi. CMT also provides a means to discover and work around cycles in the package dependencies[2], so each package will only be visited once. Any cycles will be arbitrarily broken so the Pacman files do not contain cyclic dependencies.

A special package to perform post-installation setup of the software is prepared. This does things like fixing location-dependent configuration files and providing the user environment.

Most releases have some problems to work around. The philosophy is to deal with these when the kit is created, so that the procedure to install a kit remains independent of the release as far as possible.

A typical kit size (compressed) is about 0.75 GB.

4.6. Installing the kit

The procedure for the end user or remote site administrator is straightforward.

- Install Pacman, if not already installed.
- Install prerequisite software. On RedHat 7.3, the only prerequisite not typically installed is the compiler gcc-3.2 and Java SDK 1.4.1
- Choose a directory for the installation

[2] Cycles in the package dependencies may be considered to be a flaw in the software, but it is nonetheless an example of a problem that had to be worked around to produce an installation kit.

- Choose which release to install. Available releases are listed on a web page.
- Get it with Pacman[3] by typing for example:

```
pacman -get ATLAS:AtlasRelease-6.2.0
```

To test the installation or use it, some environment variables have to be set (e.g. `PATH`, `LD_LIBRARY_PATH`). This is done with CMT, in a way that will be familiar to those who have used the ATLAS software directly from the primary site installation.

It would be perfectly possible to run automatic tests as a final step of the installation.

The full procedures and associated scripts can be found in the ATLAS Deployment package [13].

5. Conclusions and outlook

The current product broadly satisfies the requirements for run-time and developers' kits. The tools and procedures will now be improved though testing and widespread use.

It is intended to test this at the tail end of the current ATLAS data challenge, along side existing methods. It is hoped to put it into production by the end of 2003, when it should become an integral part of the ATLAS software release procedure.

Further possible enhancements include:

- Include some standard ATLAS tests at the end of the automated installation.
- Better handling of prerequisite software and platform compatibility checks. The EDG WP4 configuration management task [14] may be helpful for this.
- A full source kit distribution, so the software can build from scratch.
- The automated procedure has the potential to work with an installation on demand mechanism.
- The ATLAS software may be divided into sub-projects (e.g. simulation, reconstruction, analysis, external), in which case it would be straightforward to provide entry points to the same Pacman cache to download only the packages needed for one of these sub-projects.
- The relationship of this project to LCG/EDG/iVDGL GLUE [7] will be explored.

Finally, it is noted that version 3 of Pacman will soon be released, with many new features that this project could benefit from.

[3] It is intended to include the ATLAS cache in the registry of predefined caches whose location is known by Pacman; otherwise a full URL to the cache should be given.

6. References

[1] ATLAS experiment http://atlas.ch/

[2] GridPP ATLAS introduction http://www.gridpp.ac.uk/atlas/ATLASIntroduction.htm

[3] F. Brochu et al., EDG integration and validation in the framework of ATLAS Data Challenges', these proceedings.

[4] G. Poulard for the ATLAS Collaboration, ATLAS Data Challenges, Paper 541, CHEP03.

[5] ATLAS offline software http://atlas.web.cern.ch/Atlas/GROUPS/SOFTWARE/OO/

[6] A. de Salvo, ATLAS RPM kits https://classis01.roma1.infn.it/atlas-farm/atlas-kit/RPMKIT-install.html

[7] O. Barring et al, A GLUE Meta-Packaging proposal for Grid Middleware and Applications http://www.griphyn.org/documents/document_server/show_docs.php?category=ext&id=523

[8] CMT http://www.cmtsite.org/

[9] Pacman http://physics.bu.edu/~youssef/pacman/

[10] RPM www.rpm.org

[11] Tar http://www.gnu.org/software/tar/tar.html

[12] Gaudi http://proj-gaudi.web.cern.ch/proj-gaudi/

[13] ATLAS software deployment tools and procedures http://atlas-sw.cern.ch/cgi-bin/cvsweb.cgi/offline/Deployment/

[14] European Data Grid, Work Package 4: Fabric Management, Configuration Management task. http://hep-proj-grid-fabric.web.cern.ch/hep-proj-grid-fabric/Tasks/Configuration/

7. Figures

```
package AthExHelloWorld
author Paolo Calafiura Paolo.Calafiura@cern.ch
use AtlasPolicy    AtlasPolicy-01-*
use GaudiInterface GaudiInterface-01-* External
library AthExHelloWorld *.cxx -s=components *.cxx
apply_pattern component_library
apply_pattern declare_runtime
```

Figure 1: sample CMT requirements file for a package 'AthExHelloWorld'.

```
description='Package AthExHelloWorld-00-01-04 in release 6.2.0'
url='http://atlas.web.cern.ch/Atlas/GROUPS/SOFTWARE/OO/Development/'
source = '../dist'
download = { '*':'AthExHelloWorld-6.2.0.tar.gz' }
depends = [  'AtlasPolicy-6.2.0' , 'GaudiInterface-6.2.0'  ]
```

Figure 2: sample Pacman file for a package ‘AthExHelloWorld’.