Evaluation of the CMT and SCRAM Software Configuration, Build and Release Management Tools

Alex Undrus

Brookhaven National Laboratory, USA (ATLAS)

Ianna Osborne

Northeastern University, Boston, USA (CMS)

This document summarises an evaluation of two software release tools CMT and SCRAM which have been identified by the "Software Process RTAG" as possible candidates for a common tool supported by LCG. The evaluation of SCRAM has been carried out by a CMT expert, while CMT has been evaluated by a SCRAM expert. The main goal of the evaluation was to compare and contrast the tools behaviour in a real environment by using the tools with a real software project. As a part of the exercise several packages of the ATLAS core software (CMT-based) have been configured and released using SCRAM. Similarly, a SCRAM-based CMS software project IGUANA has been configured and released using CMT. The evaluation has been done for SCRAM version V0_19_3 and CMT version v1r12. Based on the results of the evaluation, SCRAM is proposed as a common tool for LCG project, by both authors.

I. INTRODUCTION

The web page with CMT/SCRAM evaluation reports is set at

`http://www.usatlas.bnl.gov/computing/software/cmt_scram.html`

This document summarises the evaluation of two release tools: SCRAM (version V0_19_3) and CMT (version v1r12). The evaluation applied to core software of the ATLAS experiment and the CMS software project IGUANA.

CMT is an official ATLAS and LHCb software release tool. SCRAM is a CMS release tool. For comparison with CMT, some ATLAS software was built with SCRAM: the ATLAS/LHCb framework package Gaudi (version 0.9.1) and a part of ATLAS Software Release 3.2.0 sufficient to run the simple *HelloWorld* example. The ATLAS software built with SCRAM can be found at

`/afs/cern.ch/atlas/software/dist/nightlies/SCRAM_tests/Gaudi_0_9_1`

`/afs/cern.ch/atlas/software/dist/nightlies/SCRAM_tests/AthenaExamples`

The HelloWorld example can be run from

`/afs/cern.ch/atlas/software/dist/nightlies/SCRAM_tests/AthenaExamples/run`

with

```
source environ_setting.csh
athena HelloWorldOptions.txt
```

where environ_setting.csh is a script that installs few environment variables, athena is the ATLAS framework executable (built with SCRAM), and HelloWorldOptions.txt is the file with options for athena job. The same software built by CMT is located at

`/afs/cern.ch/atlas/offline/external/Gaudi/0.9.1`

`/afs/cern.ch/atlas/software/dist/3.2.0` (full ATLAS release)

IGUANA project released with CMT can be found at

`/afs/cern.ch/user/y/yana/work/CMT/IGUANA`

IGUANA project released with SCRAM

`/afs/cern.ch/cms/Releases/IGUANA`

For more information about CMT look at

`http://www.cmtsite.org/`

SCRAM documentation can be found at

`http://cmsdoc.cern.ch/Releases/SCRAM/current/doc/html/index.html`

II. CODE MANAGEMENT

A. CMT

- CMT software release is a set of duets: "elementary" packages and their versions. The sample organisation of a CMT release is shown in Fig. 1. The internal structure of a package is restricted and requires version directories that in their turn contain sources, binaries, and the **cmt** directory that holds the configuration information needed by CMT in the form of **requirements** file. These files have a complicated syntax (30 configuration parameters with numerous options). There is no common release areas for binaries or includes packages (such as bin, lib). The compilation of each package is performed in its own subdirectory where all resulted binaries are put.

- CMT uses a CVS interface that allows to perform a limited number of cvs commands. The CMT directory structure with version directories precludes a direct CVS checkout (because there are no version directories in the CVS repository).

- CMT uses version identifiers based on CVS symbolic tags.

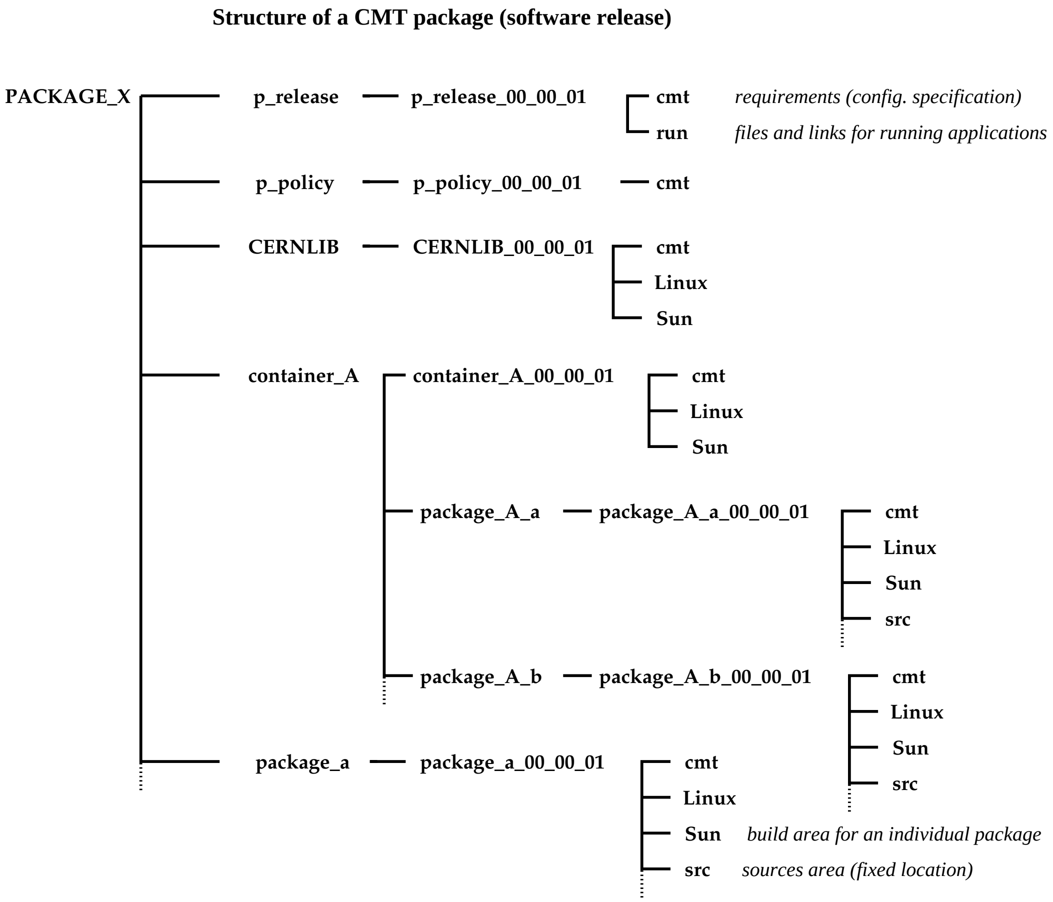

FIG. 1. Organisation of a CMT software release.

B. SCRAM

- SCRAM works with software "projects" that can include many software packages. The "project" (structure shown in Fig. 2) is a releasable unit (in sense that it is marked by a version name) and it consists of a set of documents which describe the structure and distribution of the project (in `config` directory), build configuration (in `config` directory), automatically generated external product descriptions (in `.SCRAM` directory), and project areas (libraries, binaries, includes, sources). The "project" consists of subpackages that do not have individual versions (that is each time a subpackage is changed the whole project should be released to get a new version). The organisation of subpackages is not restricted. There could be container packages. The names of source code directories are optional (CMT expects that software sources are in `src` directories). The actual compilation is performed in special `tmp` area and resulted binaries are moved to project areas (such as `bin, lib`...).

- SCRAM uses CVS, but it does not depend on it. The physical structure of a releasable unit source code is identical to its physical structure in CVS.
- SCRAM uses CVS symbolic tags for versioning.

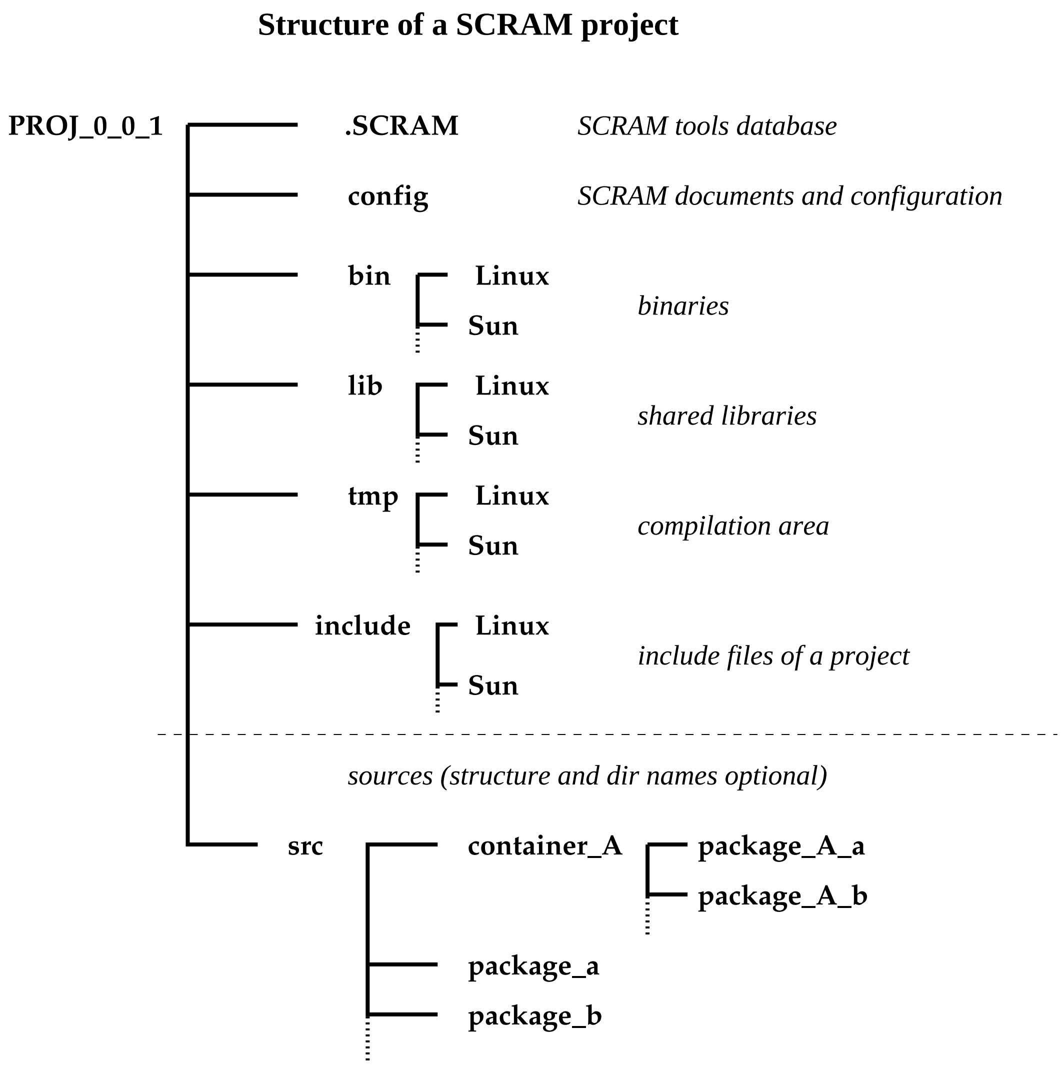

FIG. 2. Organization of a SCRAM project.

C. Comments

The organisation of a SCRAM releasable unit is more flexible. It has more transparent access to the source code stored in a CVS repository. The CMT release has fixed organisation and lacks common areas for binaries and includes.

III. BUILD SYSTEM

A. CMT

- CMT generates makefiles out of internal make fragments. Users have a limited ability to customise makefile macros in **requirements** files. There is a limited opportunity to add fragments to makefiles (so called pre-defined generic "templates" should be used). The CMT manual provides recipes for creating makefiles fragments for handling a support of languages and specialised document generators. For other non-standard actions a user may override internal CMT make fragments, but only "in the case of deep understanding of what the original CMT fragment does" (according to CMT manual).
- CMT calculates the building order from the requirements files.

B. SCRAM

- In SCRAM documents the makefiles are associated with packages. That is packages can be built with individual makefiles. SCRAM is flexible about tools used to build (e.g. could invoke preprocessing tools such as autoconf, moc, etc.). The packages individual makefiles are in fact treated as fragments of the final makefile generated by SCRAM, but the user has a freedom to create and build necessary targets.
- SCRAM compiles and links packages in the build order indicated in the project configuration file (*"config/BuildFile"*).

- SCRAM provides additional (though limited) opportunities to customise a build process using the *BuildFile* tags like appending libraries to the link line or adding include paths. A user can define a new tool or new rules for additional customisations. It appears desirable to expand *BuildFile* tags adding opportunities like changing the name of library from the default or prepending libraries to the link line.

- SCRAM mechanically adds information from the packages *BuildFiles* and generates long *make* commands with repeating compiler flags. For example, if package A uses B, and both A and B use Gaudi, then SCRAM generates *make* commands with two sets of Gaudi include and library paths. It seems desirable to filter out repeating flags (it has been implemented in SCRAM V0_19_4).

C. Comments

Both tools use make. Both tools do not reveal cyclic dependencies. Customisation of builds is much easier with SCRAM.

IV. CONFIGURATION MANAGEMENT

A. CMT

- The full set of software packages of the experiment (such as ATLAS) is naturally organised in the form of one Software Release. A special glue package describes the structure of a Software Release and other glue packages provide connections to external software.

- External tools are described in "glue packages" that contain configuration specifications in its requirements files. This substantially increases the number of packages. About 50% packages of CMT-style Gaudi are descriptions of external packages (such as CERNLIB, ROOT...). CMT does not check the availability of external tools, the problems are revealed at compilation or runtime phases.

- Version defined in the requirement file may or may not correspond to the actual version of the tool.

B. SCRAM

- Consistent configuration of SCRAM-based projects and external software packages is defined by an XML-like description (CMSConfiguration), where each project or package is considered as a tool and is described in a separate document.
- External tools are described in special documents that contain information on location of libraries, binaries, include files; library names and link ordering; dependencies; platform specifics; documentation links. The tool documents are downloaded simultaneously with the project sources.
- Once the tool documents are loaded with `scram setup` command it is possible to reload them by the second `scram setup -i` or to load them via a URL of the tool. An automatically generated .SCRAM directory with tools descriptions created in the user's area is not protected from the user.
- SCRAM checks the existence of external tools (directories and libraries) as defined in a project configuration. A user has an opportunity to ask SCRAM to skip the settings.

C. Comments

SCRAM promotes the configuration of experiment's software in the form of a hierarchy of projects. Projects can be separately installed and tested. CMT recommends to organise the experiment's software in one releasable unit. The mechanism for partial installation of a release (for instance when a user wants to install reconstruction related software only and does not want simulation related software) is not provided by CMT.

V. RELEASE AND DISTRIBUTION

A. CMT

- CMT installs soft links in the build area of a package to shared libraries of other packages and data files in the special *run* directory. Executables are supposed to be run from this *run* directory. When dealing with large projects, CMT can create very long LD_LIBRARY_PATH with paths to library files that are dispersed in the individual package areas (this creates problems for *tcsh* users because this shell limits the length of environmental variables to 1 - 4 kB).

B. SCRAM

- SCRAM has extremely useful web interface that allows to download the project sources and configuration with single click. Download without web browser is also possible.
- SCRAM suggests runtime environment settings for released or developer project. Since all libraries and binaries of a project are located in common "bin" and "lib" areas the installation of a runtime environment is not a problem.

C. Comments

Both tools allow site specific installation (that is environment is automatically tuned for different computing facilities). Information about binaries downloads is not immediately available for both SCRAM and CMT.

VI. DOCUMENTATION AND EASE OF USE

A. CMT

- Complete but complicated documentation, lacks index and quick start section.
- Tutorials are available.

B. SCRAM

- SCRAM documentation is poor. Many details and some command options are not described, though available through "scram help".

C. Comments

SCRAM is much easier to learn and use providing that a SCRAM expert is available for help (important details are missed in documentation). The complexity of SCRAM features available to a user always matches the user's experience with SCRAM.

VII. CONCLUSION

Both SCRAM and CMT are working release management tools. SCRAM is based on PERL scripts and, while it has poor documentation, those scripts are possible to read (and even adapt) for learning purposes while CMT is a large C++ package that can be changed only by a very experienced person. SCRAM allows to define more flexible directory structure of a project. The structure of a SCRAM-based project is defined from the top to the bottom and allows to avoid repetitive configuration information. It doesn't need to have versions directories and it can have common *bin, lib, include* areas. CMT maintains packages in more sophisticated way trying to handle consistently their declared relationships to each other through a version identification model. The penalty is the numerous *requirements* files with complicated syntax and "glue packages" that contain no source code, but description of the package dependencies. CMT fully generates environment and makefiles for the packages from the *requirements* files.

Both authors came to a conclusion that SCRAM is easier to learn and use, and it is more flexible than CMT in definition and configuration of a releasable unit. SCRAM has more convenient system of external tools description and it checks their availability. Though, the SCRAM build system has some limitations (for instance, one library per package), it is considerably easier to customise than the CMT one. Release management with SCRAM is more efficient. Both authors recommend SCRAM as a configuration, build, and release tool for LCG project.