BSM: Bundled Software Manager and the application for CEPC

Xianghu Zhao[1,*], *Manqi* Ruan[1], *Gang* Li[1], and *Xiaomei* Zhang[1]

[1]Institute of High Energy Physics, 19B Yuquan Road, Beijing 100049, China

Abstract. The Circular Electron Positron Collider (CEPC) is designed as a future Higgs Factory. Like most high energy physics experiments, the offline software consists of many packages. BSM (Bundled Software Manager) is thus created in order to simplify the deployment and usage of software which has many packages and dependencies. BSM utilizes Git as the software release repository. The details of software are defined in the Git repository including installation instructions, environment, dependencies, etc. Commands are supported for various shells. Json output and python API are also available for advanced development. The installation of each package could be configured separately and extended with customized handlers. BSM manages the environment variables and the version switching is easy. It also has fine environment control on a single package. Users can also define their own packages easily and these packages will be managed by BSM with simple configuration. CEPC software has already set up the deployment procedure with BSM. And BSM is also designed with flexibility to create different applications other than CEPC software. It is suitable for the projects including a lot of packages and it is safe for different BSM applications to coexist with each other under proper configuration.

1 Introduction

The Circular Electron Positron Collider (CEPC) is proposed and designed to be a Higgs factory [1, 2]. The CEPC offline software includes simulation, digitization, reconstruction, analysis chain and DAQ software, which has already been built for detector and physics study [3, 4].

The full CEPC offline software contains around 50 packages currently, where the number would increase with further researches. Besides the physics software packages, the building tools and external libraries should also be included in the software.

The stability for the software release could be another issue. Unlike single package, a normal software release includes many packages, all of them having their own versions. A full release version must deal with all the packages and their versions altogether.

Therefore, there should be some kind of management tool to deal with these issues. Many tools could do similar jobs, like CMT [5], CMake, Spack, Environment Modules, etc. However, they cannot meet all the requirements for the CEPC software management.

*e-mail: zhaoxh@ihep.ac.cn

Bundled Software Manager (BSM) was created for these objectives [6]. It is designed as a general-purpose tool which could be easily applied for different projects. It has great flexibility and extendibility, and it does not restrict the package source or building tools. It simplifies the work for physics users, software developers and release maintainers, and it could help with the process of software release, installation and package development.

BSM uses Git as the software repository. The detailed behavior of the application could be customized with YAML configurations and Python handlers.

2 Architecture of BSM

The most important job for BSM is setting the environment variables. Figure 1 shows the main components and the basic workflow that generate the final environment.

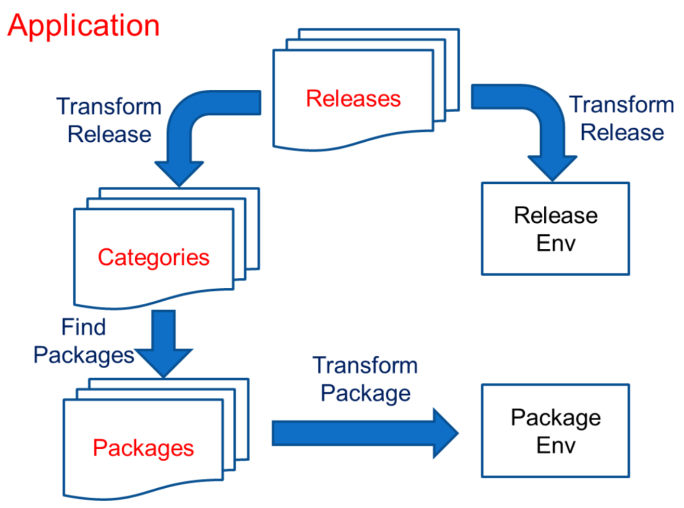

Figure 1. BSM architecture and workflow

2.1 Concepts

BSM defines several concepts in order to organize the content of software. These concepts would help to understand the architecture of BSM.

2.1.1 Application

An application includes the full set of the software, which defines the following properties:

- The basic information about the application, including application name, description, web site, etc.
- The Git repository for hosting all the releases.
- The version pattern on how to find release versions from the tags of Git repository.
- Other configuration which could affect the behavior of the application.

2.1.2 Release

A release defines a single version of the software release, which corresponds to a Git tag in application repository with matched pattern. The publish of a release is created when a version tag is pushed to Git. The release consists of:

- The definition of categories included in the release.
- Configurations for all packages to be installed in the release.
- The installation steps which are used to describe the installation procedure.
- Global environment settings for the current release.
- Handlers for installation and other operations.

2.1.3 Category

Categories include the packages with similar properties. They are defined in the release or user configuration. Normally a category corresponds to a directory which would hold many sub-packages. Different categories could have the same packages, of a different version or not, and category priority is used to determine which package/version should be active.

2.1.4 Package

The package is the smallest unit in BSM. It should correspond to the conventional package, which may be a library, toolkit, command, etc. Each package belongs to a specified category, and has a fixed version. The package versions are defined in the release along with their dependencies. Each package has its properties and environment settings, which will be used in the installation and runtime.

2.2 Release installation

The full release installation is controlled by the single install command. The release needs to define the global installation steps and installation handlers, and then specify them in the package configurations. An optional environment check could be done before the installation starts. The installation steps among packages would automatically run in parallel in order to speed up. The dependencies are considered according to the package configuration.

The installation method is defined in the release handler, so the source of package may be hosted on HTTP server, SVN, Git, etc. Binary package extraction or building from source are both acceptable. Extra checks before or after each package installation could be added as a step in the release configuration.

2.3 Command and environment management

BSM provides a command to control the operations and environment, where the command name could be configured by application. Since the child process is not able to alter the environment of parent process, the BSM command is actually a shell function or alias. Currently supported shells include sh, bash, zsh, csh, tcsh, and more could be added if required.

The environment settings are quite flexible in BSM. They could be set for a whole release including specified categories with sub-packages. The switch between releases and environment cleaning are all controlled with single command.

2.4 Package based operations

It is common to manage individual package during development. BSM provides several package based commands which could install, create and setup a single package.

All packages belong to certain category, therefore package creation must be done under a specified category. There could be more than one version for a single package, which can be located under the same or different category. However, only one version will be active at the same time, and the versions can be switched with package command.

2.5 Transforming handler

Transforming handler written in Python is a powerful method to modify release and package configurations on the fly. There are two kinds of transforming handlers for release and package individually, which can be used to implement some advanced features. Transforming handler could accept options from user command, and change the behavior of release or package according to user input. It can also simplify the package configuration, and create environment dynamically.

3 Application for CEPC offline software

BSM is originally designed for CEPC offline software, so it should be the most comprehensive use case currently. The CEPC software creates release from cepc-release repository, and defines the cepcenv command [7].

3.1 Installation steps

Most CEPC software packages are built from source with CMake or GNU Make, and sources are located from various repositories. Some packages need patch in order to compile successfully. The intermediate files from compiling should be cleaned because of the large size.

CEPC release defines the following steps: download, extract, patch, pre_build, configure, compile, install, post_build and clean. Sources or data files are downloaded from http server, SVN or Git repository. Files are extracted in case it is in archive format. Patches are applied on demand. The build process is divided into three steps: configure, compile and install. The cleaning of intermediate files is optional.

The check for system requirements is applied before installation starts. In case anything is missing, useful information will be provided for user.

3.2 Categories

There are 8 categories predefined by the CEPC release. The external, cepcsoft and data categories will be installed in the release installation. The work and work_ver categories are used by physics user.

- not_used. Any package with this category set will be ignored directly.
- system. These system packages must be pre-installed before the CEPC software installation.
- external. This category includes external tools and libraries which are used by CEPC software. They could be shared by different release versions.
- cepcsoft. The core CEPC software category.

- data. Common data packages which could be shared by all versions and platforms.
- standby. The packages under this category directory are installed individually. They are used as extra packages or extra package versions which are not included in the standard release. They are not installed or loaded with release automatically, but could be loaded with package command manually.
- work. It is provided to the physics user to develop their own packages. Packages under this category are loaded automatically and have higher priority than the standard release packages. This category is shared among release versions, which means there is no need to create a new work directory for each release. The work directory could be configured with option.
- work_ver. It is similar to work category, but has version directory level for each package.

3.3 Platforms

The platform for CEPC software is defined by the system architecture, operation system type and compiler version. Installation from different platforms will be placed in separate directories. Then they could be all installed on the same shared file system and automatically chosen according to the platform. The platform could also be specified in the options in case it is not correctly detected.

3.4 Options and handlers

There are several options defined for installation. The download source can be changed to IHEP mirror. The number of jobs for make command can be specified by user.

Several transforming handlers are defined in the release. One handler will try to guess the version from CMakeList.txt file for work category, thus the user does not need to set the package version explicitly.

4 Conclusion

Currently the BSM is released and successfully applied for CEPC offline software, which has well simplified the software usage and development. As a general-purpose toolkit, BSM could also be used by other projects with amounts of packages, which in return will help to improve BSM itself.

Acknowledgements

References

[1] The CEPC Study Group, *CEPC Conceptual Design Report Volume I: Accelerator*, IHEP-CEPC-DR-2018-01, arXiv: 1809.00285

[2] The CEPC Study Group, *CEPC Conceptual Design Report Volume II: Physics & Detector*, IHEP-CEPC-DR-2018-02, arXiv: 1811.10545

[3] F. An et al, Chin. Phys. C **43**, 043002 (2019)

[4] M. Ruan et al, Eur. Phys. J. C **78**, 426 (2018)

[5] C. Arnault, *CMT: A Software Configuration Management Tool*, CHEP 2000 proceedings, Padova, Italy. (2000)

[6] The BSM project, https://bsmsoft.github.io/

[7] The CEPC release project, https://github.com/cepc/cepc-release