PACKAGING OF CONTROL SYSTEM SOFTWARE

K. Zagar, M. Kobal, N. Saje, A. Zagar, Cosylab, Ljubljana, Slovenia,
R. Sabjan, COBIK, Solkan, Slovenia,
F. Di Maio, D. Stepanov, ITER Organization, St. Paul lez Durance, France

Abstract

Control system software consists of several parts – the core of the control system, drivers for integration of devices, configuration for user interfaces, alarm system, etc. Once the software is developed and configured, it must be installed to computers where it runs. Usually, it is installed on an operating system whose services it needs, and also in some cases dynamically links with the libraries it provides. Operating system can be quite complex itself – for example, a typical Linux distribution consists of several thousand packages. To manage this complexity, we have decided to rely on Red Hat Package Management system (RPM) to package control system software, and also ensure it is properly installed (i.e., that dependencies are also installed, and that scripts are run after installation if any additional actions need to be performed). As dozens of RPM packages need to be prepared, we are reducing the amount of effort and improving consistency between packages through a Maven-based infrastructure that assists in packaging (e.g., automated generation of RPM SPEC files, including automated identification of dependencies). So far, we have used it to package EPICS, Control System Studio (CSS) and several device drivers. We perform extensive testing on Red Hat Enterprise Linux 5.5, but we have also verified that packaging works on CentOS and Scientific Linux. In this article, we describe in greater detail the systematic system of packaging we are using, and its particular application for the ITER CODAC Core System.

INTRODUCTION

The principal challenges of today's control systems for large experimental physics facilities are complexity and quality assurance.

By this we mean the fact that a large number of software components – executing either on the same host or in a distributed set-up – need to be integrated into a functioning whole while performing according to performance and reliability expectations. The complexity challenge stems both from the inherently distributed, large-scale nature of a control system, as well as the trends in component-based software engineering and systems engineering, where monolithic systems are giving way for those that are integrated from smaller, more manageable subsystems.

With limited development, maintenance and integration resources – in particular skilled staff – it is important that as many tasks as possible are automated, and that common problems have common solutions – i.e., that standardization takes place to the extent that it is economically feasible.

The ITER CODAC [1] control system is also facing these challenges. CODAC integrates software packages that are a product of two decades of work, and which have been developed in diverse environments by different teams. Not surprisingly, each of these packages takes a different approach on how the software is built, and what quality assurance process is in place during its release.

We have decided to standardize at least the interface with which the developer (or maintainer) interacts with the build system. To achieve this, we have wrapped the diverse approaches and technologies for building (Makefile [2], Ant [3], shell scripts, Eclipse builder [4], etc.) into one tool. Since many software packages share the same approach (e.g., EPICS base [5] and all of its extensions rely on Makefile, while Control System Studio [6] and all of its plug-ins rely on Eclipse), we were looking for a way to re-use our effort: for example, specify integration with the EPICS Makefile system in a single place, and "invoke" it with a one-line stanza in all software packages where it is needed.

As we are not the first to have come across this challenge, a market survey revealed that build tool frameworks already exist (for example, Maven [7] and Gradle [8]). After our evaluation, performed in late 2009, we have settled to use Maven 2 as the platform, and we have chosen to solve our challenges by implementing a plug-in for this tool.

Another challenge is managing deployment across the many hosts that will eventually constitute the control system. However, this challenge is not uncommon in the IT industry, where large corporations also have thousands of computers that need to be managed with the limited IT staff. To leverage existing solutions, ITER had decided to take an off-the-shelf approach: using Red Hat Enterprise Linux and its automated installation and update capabilities enabled with the Red Hat Satellite software [9].

Managing installation, un-installation and updating of software packages on an individual host is a rather complex task in itself. The most trivial step of it is to place the files constituting a software package (executables, scripts, configuration files and data files) to the right places in the file system. As un-installation and update need to be able to clean-up those files, meta-data must be associated with each file to specify which software package had installed it. Installation/un-installation might require that some actions are taken (e.g., adaptation of configuration files of other software components, creation of database schemas, population of databases, etc.). And finally, software package might have dependencies, and other software might depend on: installing a package might thus have a precondition that

```
# Spec file for package rf-ich-sample-MCioc
# Generated by the codac-packager Maven plugin.
# Date: Fri Sep 30 13:32:55 CEST 2011 ...

%define unit_version_full %{codac_version_full}.v1.0a1

Name:    %{codac_rpm_prefix}-rf-ich-sample-MCioc
Version: %{codac_version_full}.v1.0a1
...

Requires:        %get_current codac-core-3.0-epics-autosave

%description
Input (a snippet from Maven pom.xml):

%install
sed -r -i 's#epicsEnvSet\("TOP"\s*\,\s*".*?"\)#epicsEnvSet\("TOP","/opt/codac-3.0/apps/rf-ich-sample"\)#g'
%{buildroot}/opt/codac-3.0/apps/rf-ich-sample/iocBoot/iocMC-ICHCore/envPaths
...

install -d %{buildroot}/etc/opt/codac-3.0/alt.d/
echo --slave \"%{_bindir}/MC-ICHCore-ioc\" \"codac-sudo-MC-ICHCore-ioc\" \"/opt/codac-3.0/bin/services/sudo-
service\" >> "%{buildroot}/etc/opt/codac-3.0/alt.d/rf-ich-sample-MCioc"
echo --slave \"%{_initrddir}/MC-ICHCore-ioc\" \"codac-srv-MC-ICHCore-ioc\" \"/opt/codac-3.0/bin/services/MC-
ICHCore-ioc\" >> "%{buildroot}/etc/opt/co

%pre
...
```

Figure 1: Example of a SPEC file that provides meta-information and build instructions for an RPM package.

other packages are installed beforehand, and uninstalling it may have a consequence that those depending on it should be uninstalled as well.

This problem, too, has already been solved by the IT community. On Linux platforms, the Debian package management (APT/DEB) and Red Hat Package management (YUM/RPM) are commonplace. As ITER has chosen Red Hat Enterprise Linux as the operating system, we have opted for the YUM/RPM technology.

To provide a RPM, the developer must provide a so-called SPEC file. The SPEC file contains (see Figure 1):

- meta-information about the package (name, version, description, etc.),
- instructions in form of an executable script on how to build the package (unpack the sources, run configure/make or other tools, etc.)
- which files to package, and what default permissions to assign to them
- the scripts to execute before and after installation, and before and after un-installation.

CONSTITUENTS OF A CONTROL SYSTEM

In case of ITER CODAC, the control system consists of the following kinds of software packages:

- EPICS IOC applications.
- Configuration files for the BEAST alarm server.
- Configuration files for the BEAUTY archiving system.
- Kernel modules, e.g., for implementation of kernel-mode device drivers.
- User-mode device drivers and libraries.

The EPICS IOC applications are standard EPICS applications, built with the EPICS' Makefile system. They consist of the binary compiled for the target platform, the `st.cmd` start-up script, and EPICS database files.

These files are packaged in an RPM package, and an `init.d` script is automatically generated that allows for starting up and shutting down of the IOC as a system service. The `init.d` script also provides a console through which developers and maintainers can access the EPICS shell of the IOC process (via the `screen` tool).

For security reasons, it is not advisable to run services as the `root` user. Therefore, a system user called "`codac`" is provided, and all services run under that account. This raises some issues with permissions (e.g., the `codac` user by default doesn't have permissions to interact with kernel-mode device drivers, nor does it have permissions to set its real-time attributes such as scheduling priority and CPU affinity). The packaging ensures also that these permissions issues are properly addressed.

Packaging of configuration files for BEAST and BEAUTY involves putting the configuration files in the RPM package, and running the database import tools upon installation of the RPM to populate the BEAST and BEAUTY configuration databases with their content. In CODAC, the content of these configuration files is automatically generated by the SDD tools [1], thus they are in-sync with the contents of the EPICS configuration database.

Kernel modules are built with standard tools for kernel modules. Currently, kernel modules can be built for two targets:

- A regular kernel.
- A real-time kernel.

```
<package name="MCioc">
  <include name="MC-ICHCore" type="ioc"/>
  <include type="boy" file="*"/>
  <include type="databrowser" file="power.plt"/>
</package>
```

Figure 2: Excerpt from Maven's POM XML file responsible for generating RPM SPEC file from Figure 1.

For each build, a separate RPM package is provided, which then has its dependency set as required (either to the `kernel` or to the `kernel-rt` package).

THE MAVEN PLUGIN

The packaging of control system's constituents described in the previous section into RPMs is performed by the a Maven plugin we have developed.

The Maven plugin is designed to execute the packaging task during the "`package`" phase of its lifecycle – i.e., after compilation and testing, but before installation and deployment.

The description of the packaging is very concise. Figure 2 shows an example of the Maven configuration that results in RPM SPEC file shown in Figure 1.

The RPMs thus produced allow installation of several versions of the ITER CODAC system simultaneously. E.g., version 2.1 and 3.0 can be installed at the same time, with the first residing in `/opt/codac-2.1` and the latter in `/opt/codac-3.0`. The developer can then switch between the two versions with a command `codac-version`, which redirects all the softlinks from the system's paths accordingly (via the *alternatives* system) and reconfigures environment variables.

In addition, the Maven plugin provides command-line options that facilitate generation of Maven's project definition file, the `pom.xml`. Thus, even developers not familiar with syntax of this file can configure their projects to be packaged.

USAGE EXAMPLE

The developer might use the Maven-based tools as follows.

Firstly, the developer would create a *project* (a *unit* in ITER's CODAC terminology) by executing:

```
mvn iter:newunit -Dunit=my-unit
```

This results in a subdirectory *m-my-unit* (the *m-* prefix is prepended due to ITER CODAC's naming convention for units). The directory structure conformant to CODAC's standards, and Maven's `pom.xml` file, are also created by this command.

The following sequence of commands creates an EPICS application, and then configures an IOC process to run that application. The type of the application here is "psh", referring to ITER CODAC's *Plant System Host*:

```
cd m-my-unit
mvn iter:newapp \
    -Dapp=PlantSystemHost \
    -Dtype=psh
mvn iter:newioc \
    -Dioc=PlantSystemHost \
    -Dapp=PlantSystemHost \
    -Dtype=psh
```

Now, the RPM can already be prepared by executing:

```
mvn package
```

It is also possible to conveniently start the resulting IOC:

```
mvn iter:run
```

CONCLUSION

The Maven-based tools that have been developed to facilitate the development process greatly simplify development of software for the control system – from input/output controller processes to extensions of the operator's graphical user interfaces.

The interface provided to developers is simplified so that even those who had no prior exposure to EPICS, Maven or RPM are able to create new EPICS-based, CODAC-compliant projects, build them, and ensure that results are packaged in an installable RPM.

While the tools have been developed for ITER, they can be adapted to other projects as well. For example, currently an effort to adapt ITER CODAC for the needs of the European Spallation Source's control system.

REFERENCES

[1] F. Di Maio et al., "The CODAC Software Distribution for the ITER Plant Systems", ICALEPCS'11, Grenoble, France.
[2] GNU Make; http://www.gnu.org/s/make/.
[3] Apache Ant; http://ant.apache.org/.
[4] Eclipse; http://www.eclipse.org/.
[5] Experimental Physics and Industrial Control System; http://www.aps.anl.gov/epics/.
[6] Control System Studio; http://cs-studio.sourceforge.net/.
[7] Apache Maven; http://maven.apache.org/.
[8] Gradle; http://www.gradle.org/.
[9] Red Hat: Red Hat Network Satellite; http://www.redhat.com/red_hat_network/.