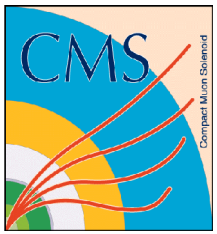
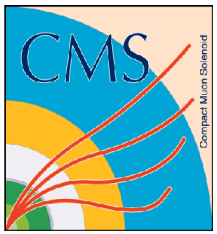


CMS software distribution

S. Argiro', Cern and INFN



Introduction

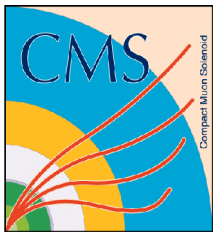
- As for almost everything else in CMS sofware,
  we are revising the distribution system.
- The current distribution system(s) have been used succesfully,
  but they start to show their shortcomings
- We will describe the current system and the plans for improvement

Current system

Two ways of distributing CMS software

1. **RPM** for development and analysis
2. **DAR** for MC production

Distribution **A**fter **R**elease :
reproduces the runtime environment.
Binaries, libraries and configuration files are packed into a tar file and shipped to remote sites.
When installing several versions, duplication occurs.
Especially insteresting for running on opportunistic resources.



Current system - RpmGen/XCMSI

RpmGen is a tool to pack a reference installation of CMS software into RPMs.

It queries SCRAM for the list of external tools used for the project that is to be distributed, their versions and locations.

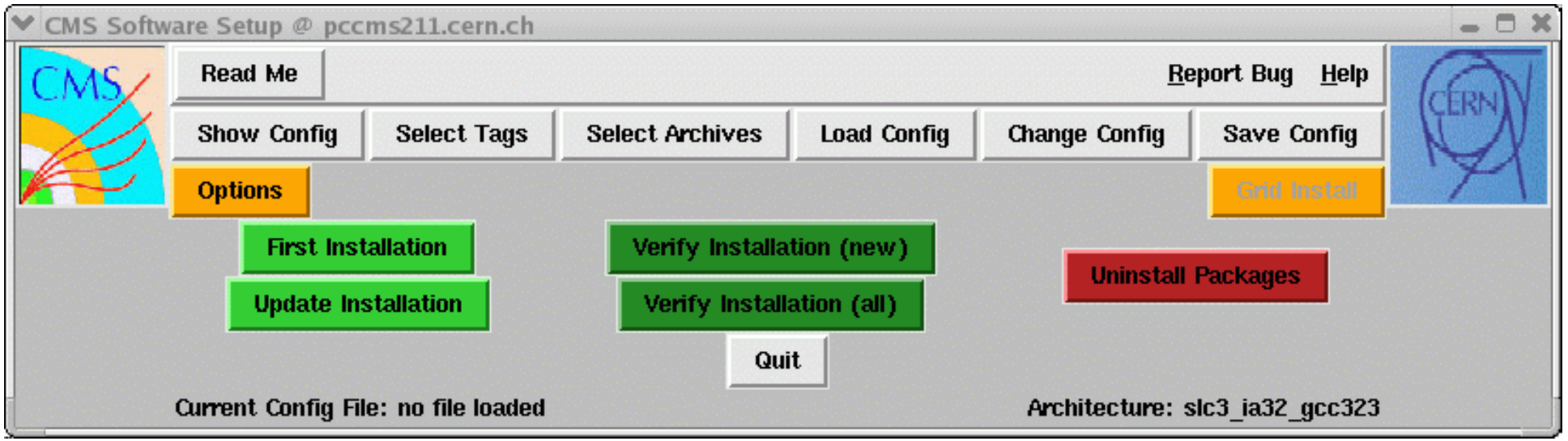
- Dependencies are handled correctly when provided by SCRAM.
- A separate RPM database is used (independent from the system db).
- Version and platform encoded in the name of the RPM
  -> support for multiple versions
- Decoupling from system RPMs.

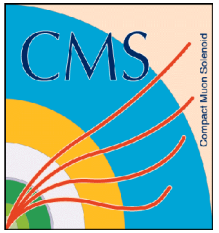
- RPM repository

Current system - RpmGen/XCMSI

XCMSI provides machinery on top of RpmGen.

- Command line tools for download, install, verify (as much as possible) packages.
- Generation of jobs to install packages on the Grid.
- GUI
- Supports installation on a stand-alone machine (laptops)
- Work in progress on automatic updates.





Current system - Problems

1. The distribution is based upon a reference installation (CERN AFS). That reference installation is different from all other installations.

2. The release manager needs to release first (and install at CERN) before he can start packaging. In practice, when a release of the software is ready, people have to wait until the packages are ready in order to start using it.

3. The AFS release (installation) area is (unfortunately) often tweaked, especially for CMS tools upon which CMSSW depends on.
The CMSSW release team does not have total control of the installation area, and, therefore, of the distribution pack.

Current system - Problems /2

4. Often difficult to eradicate AFS binding from packages.

-> things work at CERN but not other places.

5. RPM db corruptions

6. Size: 2 Gb of RPMs, 7 Gb unpacked !

5.3 lcg (probably not optimized packaging, eg. sources included)

0.8 CmsReleases (CMSSW, IGUANA, etc..)

0.8 external (cms specific externals)

Current system - Problems /3

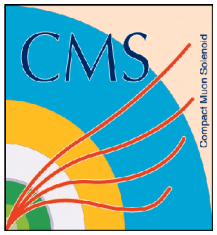
Grid installation problems:

1. We send the installation job as Cms Sw manager. This job has no special priority, can sit in queue for long times and suffers the same problems as any wimpy job.

2. The job often fails for local configuration problems: we are implementing preliminary checks.

3. Mostly, the Sw is installed on a NFS server serving several WNs. Several jobs could start in parallel and try to load the same set of hundreds of libraries -> is this a scalable solution ?

4. Availability of Sw at a given site is advertised via glue-schema, but the different OS flavours is not taken into account. In particular, the OS tag is not uniform (SLC, SL, ScientificLinux, etc)



New System - RPM generation from source

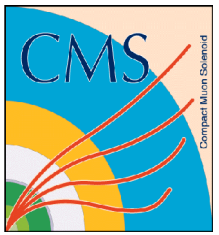
We are in the process of implementing a different RPM generation strategy. RPMs are generated from sources.

Advantages:

1. The packaging can become part of the release procedure
2. Gain complete control of the distribution pack
3. Facilitate porting
4. Configure externals as needed

We want to build all RPMs from source and store them in a repository. When there is a change in the project configuration, the RPM for the new (new version) of the tool is generated.

Installation at CERN of CMS project and external should be done using the generated RPMs.
Packaging of the distribution kit becomes part of the release procedure.



New System - RPM generation from source

The system consists of a single bash script and a set of .spec templates.

Some functionality on top of RPM is provided:

1. encoding the rpm name starting from the cathegory/version
2. download the sources from the “Source” directive
3. use a local RPM db
4. setup some RPM scripting
5. sourcing the environment to build packages
6. Rebuild all dependencies if necessary

New System - spec template example

```
### RPM external boost 1.32.0

%define boostver _%(echo %v | tr . _)
Requires: boost-build python
Source: http://dl.sourceforge.net/sourceforge/%n/%{n}%{boostver}.tar.gz

%prep
%setup -n %{n}%{boostver}
#%patch
%build

PR="PYTHON_ROOT=$PYTHON_ROOT"
PV="PYTHON_VERSION=$(echo $PYTHON_VERSION | sed 's/\.[0-9]*$//')"
case $(uname) in
  Darwin )  bjam -s$PR -s$PV -sTOOLS=darwin || true ;;
  * )       bjam -s$PR -s$PV -sTOOLS=gcc ;;
esac
%install

....
```

will generate
external+boost+1.32.0-1-1.slc3_ia32_gcc323.rpm

variables taken from package/init.sh
sourced automatically from requirements

system will download sources if necessary

New System - spec template example

```
### RPM lcg seal-tool-conf 40
Source0: none
Requires: gcc
Requires: pcre
Requires: zlib
Requires: bz2lib
Requires: uuid
        […]

%prep
%build
(echo "ARCHITECTURE:%{cmsplatf}"
 echo "SCRAM_BASEPATH:%{instroot}/external"
%if "%{?use_system_gcc:set}" == "set"
        echo "TOOL:gcc3:"
        echo "   +GCC_BASE:/none"
        echo "   +CC:$(which gcc)"
        echo "   +CXX:$(which c++)"
        echo "   +PATH:/none"   # useless, toolbox says value=""
        echo "   +LD_LIBRARY_PATH:/none" # useless, toolbox says
value=""
        echo "TOOL:g77gcc3:"
        echo "   +FC:$(which g77)"
%else
        [...]
) > tools.conf
%install
mkdir %i/configurations/
cp tools.conf %i/configurations/tools-STANDALONE.conf
```

example for scram managed project:
Setting up tools

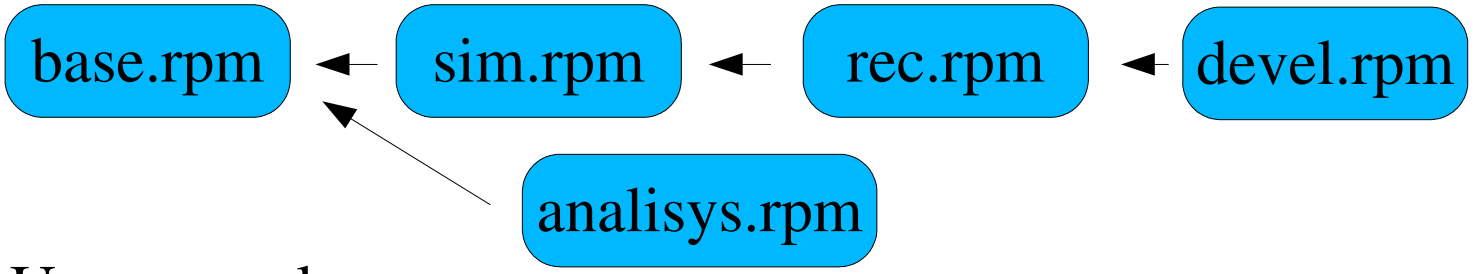
New System - spec template example

```
### RPM lcg seal SEAL_1_7_8
Requires: seal-tool-conf
%define confversion 40
%define toolconf ${SEAL_TOOL_CONF_ROOT}/configurations/tools-
STANDALONE.conf
Patch0: SEAL
%define patchsrc %%patch0
%define cvsdir %(echo %n | tr 'a-z' 'A-Z')
%define cvsserver %(echo %n | tr 'A-Z' 'a-z')
%define buildtarget release
## IMPORT lcg-scram-build
## IMPORT scram-build
%define conflevel       %{nil}
```

boilerplate scram build steps

New System - wishes

Incremental, modular distribution



Use example:

```
[user@laptop] cmsi cmssw-analysis.rpm
```

Resolves deps, installs.

Turns out to be a challenging task, since the system was not designed for that from the start.
In principle, the plugin architecture should help.

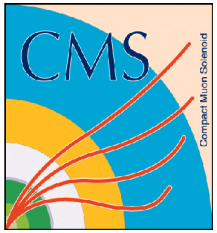
Miscellanea/Considerations

1. If all external packages were built with the same tool, life would be much easier (most of open source relies on automake, curiously Hep software does not, in general. And it lacks configurability). Some of the build systems: SCRAM (v0,v1), make, automake, scons, bjam.

2. We often feel the need of configuring external packages, but this cannot be done without hacks.
Example : POOL depends on vdt, but we do not use the part of POOL that depends on vdt... We do not want to distribute 280 Mb of vdt !! We have to hack the build of POOL.

3. There are **so many packages** and dependencies are so deep, that we think that packing our own distribution is the only way to keep things under control. Too many variables at play.

4. Many packages have functionalities that superimpose (example: how many externals in CMSSW provide matrix manipulation ?)



Conclusions

1. Distributing CMS software is a headache. From our side, deps could be simplified *(example: it's nice to have valgrind in your path, but should not be strictly required when installing CMSSW. Duplication).*

2. We believe the packaging of a distribution kit has to be specific to the experiment

3. What we would like from LCG

a. Better configurability/modularity of projects

Examples: - being able to build POOL w/o vdt

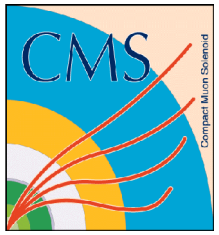
- being able to build vdt-client libraries only

Disclaimer: maybe this is already possible, but we don't know.

b. Ideally, one build system

c. A page of instructions on how to build AA software.

d. Continue the builds on supported platform of AA software



current CMSSW dependencies

jcompiler 1.4.2
graphviz 1.9
iguana IGUANA_6_7_1
meschach 1.0
doxygen 1.4.1
icutils 2.4
fed9uutils 2.4.1
cmkin CMKIN_6_0_0
pythia6_227 1.0
pythia pythia6_227
genser GENSER_1_2_1
frontier_client 2.4.1_cms
x11 R6
qutexmlrpc 0.2_qt334
hippodraw 1.15.6.1
openssl 0.9.7d
simage 1.6.1
jpeg 6b
coin3d 2.4.4
openinventor coin3d
opengl XFree4.2
soqt 1.3.0p_coin3d244_qt334
qt 3.3.4
cppunit 1.10.2
rulechecker CMS2.6_p1
dcap 1.2.35
pcre 4.4
bz2lib 1.0.2
boost_python 1.32.0_python242
gccxml 0.6.0_patch3
mysql 4.0.24
mysqlpp 1.7.31_mysql.4.0.24
myodbc 3.51.06_mysql.4.0.24
unixodbc 2.2.6
xdaq XDAQ_3_4
ignominy IGNOMINY_3_8_0
relax RELAX_0_1_0
seal SEAL_1_8_0
pool POOL_2_3_1
coral CORAL_1_2_2
lcgaa LCG_41
ccompiler 3.2.3
f77compiler 3.2.3
cxxcompiler 3.2.3
oracle 10.1.0.3-1
sqlite 3.2.8
gsl 1.5
uuid 1.38
rootinteractive 5.08.00b
rootcore 5.08.00b
rootrflx 5.08.00b
root 5.08.00b
xerces-c 2.7.0
boost 1.32.0_python242
python 2.4.2
sockets 1.0
zlib 1.1.4
rfio 2004
aida 3.2.1
clhep 1.9.2.2
heppdt 2.02.02
hepmc 1.26
geant4 7.1.clhep1922.p