Software Development and Life Cycle

Reiner Hauser

Many thanks to:

Luciano Orsini, Andrea Petrucci (CMS) Sylvain Chapeland, Barthelemy von Haller (ALICE), Marco Clemencic (LHCb),

Why this talk ?

- Almost everything beyond the detector read-out buffers consists of commercial hardware and software:
  - X86 based PCs
  - Linux as standard operating system
    - Occasional custom driver...
  - (mostly) standard network technologies and protocols.
  - Open source and widely available compilers (GCC)

➔ The DAQ/HLT area is basically a large software project.

- How do we organize, develop, debug, deploy this software in our various experiments ?

Software Life Cycle

- Buzzword list
  - Analysis
  - Requirements
  - Design
  - Implementation
  - Testing, Integration
  - Deployment
  - Maintenance

We are all here and possibly Start another major cycle in LS1

Online Software

- Emphasis here on “online” software (but mostly ignoring details of trigger)
- Each experiment structures their software in a different way
  - How many different 'projects' are there ?
  - Are they rather integrated or strictly separated ?
  - Even the choice of main language can differ by project.
- Rather interesting to sort this out at the beginning of our discussions...

ALICE

- DAQ+Run Control (C, Tcl/Tk, php+mySQL)
  - Plus “detector algorithms” for subdetectors
  - Static linking is used to avoid runtime dependencies – different projects can use their preferred version.
- DQM (C++, uses offline code – AliRoot)
- HLT (remember: running before event building – quite different from other experiments)
- Strong emphasis on logbook for sharing and presenting monitoring data.

ATLAS

- All online and off-line software is based on a common set of external software (LCGCMT), compiler version/flags.
- Structured into logical projects with a small common base
    - TDAQ project (C++, Java, Python)
    - Offline project (including trigger code) (C++, Python, Java for event display)
    - AtlasHLT project brings the two together.
- Releases of TDAQ and off-line are done separately but coordinated.
    - Typically all projects are rebuild together.

CMS

- RCMS – run control (Java)
- XDAQ – data acquisition
  - Dynamically links to off-line CMSSW for the filtering algorithms.
  - Implies common externals, compiler versions etc.
- Online software talks to each other using web services (SOAP).
- Plan is to have a stricter separation between DAQ and filter tasks after LS1.

LHCb

- Run control tightly integrated with PVSS
- All other software organized as a set of projects, even using the same framework (Gaudi) for online and offline software. (C++, Python); can talk to DIM, SMI++ etc.
- Projects can be build separately, e.g. stable base components like Gaudi are changing less frequently than analysis projects.

Environment for Software

	OS****	Main Languages	HW	Cross compilation
ALICE	SLC5	C, C++	x86	
ATLAS	SLC5	C++, Java, Python	x86	PowerPC*
CMS	SLC5, (MacOS)	C++, Java	x86	
LHCb	SLC5, (Windows***)	C++, C, Python*****	x86	ARM**

Plus: whatever web browser you use...including its JavaScript interpreter

*** detector software, also part of TDAQ project**
**** Initial investigation**
***** until 2011, mostly for developers (Visual Studio); may come back.**
****** everybody plans to move to SLC6**
******* framework configuration is done in Python.**

GCC is the main compiler, icc, clang are tested by various experiments

Version Control

- ALICE
  - Subversion, CVS (self hosted)
- ATLAS
  - Subversion (different off-line and TDAQ repos)
- CMS
  - Subversion (multiple repos), some subdetectors on CVS
- LHCb
  - Subversion
  - Git for Gaudi (self hosted – now there is IT service) and small independent projects.

Build Systems

- ATLAS and LHCb use CMT for the bulk of their software; detector software usually also uses this.
  - LHCb will move to CMake; there is also a discussion in ATLAS on changing the build system.
- ALICE use *make* for the DAQ software, CMake for the DQM project.
- CMS uses *make*, different from their off-line build system, *ant* for Java based project.

Nightly Builds and Testing

- ATLAS and LHCb are doing nightly builds
  - ATLAS: 2 branches right now, default + new compiler
  - LHCb: 3 nightly "slots" (head versions against stable LCG sw, against LCG nightly, etc).
- Unit tests: Some, universally not considered to be in good state: test should be written from the beginning, not added afterwards.
- Integration tests: Some, often makes more sense for framework oriented code.
  - LHCb: qm_test => ctest, nosetest
- ALICE uses a homemade continuous integration system for DQM
- ALICE and LHCb are looking into continuous integration for the future (maybe based on Jenkins: http://jenkins-ci.org)

Releases

- All groups have the concept of a (major) release
  - But different ways to handle minor release/bug fixes/patches, i.e. how to handle the maintenance of the running system.
- The differences are coming from the way a given piece of software depends on others, or has to provide binary compatibility to others.
- Release notes about major changes for users are provided.

Release Frequencies

- ALICE: few dependencies between software
  - 1-2/month for DAQ, 40-50/y for DQM, ~100/year for detector algorithms.
- ATLAS & CMS: dependency on off-line software version, provide stable binary API to detectors
  - CMS: 1-2 main releases/year, “update release” every few months.
  - ATLAS: 1 main release/year, “patches” on a per package basis as required (typically deployed ~once per week)
- LHCb: common software environment
  - e.g. framework 1/month, but on-line uses separate branch and diverged for a while at the end of running.
  - Separate 'patch project' for each major project.

Deployment

	Format	Installer	Granularity	Remote Site installation
ALICE	RPM	Sysadmin, system DB	1 RPM/DAQ 1 RPM/DQM	Yes
ATLAS	RPM	swinstaller role, private DB	1 RPM/package	Yes, also CVMFS
CMS	RPM	Sysadmin, system DB	1 RPM/package	Yes, requires root privileges.
LHCb	“tar ball”	Cronjob, non-root	Per project	Yes,going to RPM; looking at CVMFS for on-line as well

The installation method at the detector site is driven by the way the machines are setup, e.g. all with local disks => local copy of software, netbooted => installation on file server etc.

Patches and Bug Fixes

- ALICE has frequent release updates (few dependencies on other projects, no need for binary compatibility for some)
- LHCb has 'patch projects' for each major CMT project.
- CMS has 'update releases' which contain RPMs for a subset of the full 'main release'; they are binary compatible with older versions.
- ATLAS updates RPMs for single packages as required
  - Note: AFS installation can be different from P1, since patches can be cherry-picked by importance and urgency.
  - Note: HLT code uses single 'patch project ' similar to LHCb, about 1/week with occasional full release.

Issue Tracking

- ALICE: Jira (IT hosted, private instance for their own plugins, work flows), Logbook
- ATLAS: Savannah for bug reports, feature requests, patches.
  - Patches require a bug# for justification.
- CMS: Trac interfaced to Subversion
  - *Any* commit requires an open issue in Trac, either for a bug or a new feature.
- LHCb: Savannah
- All Savannah users plan to move to IT supported Jira.

Documentation

- Web, (t)wikis, EDMS, internal notes.
- doxygen for documentation generated from code (ATLAS, CMS, LHCb)
- Nightly build and test results shown on web pages.

Changes for LS1/LS2 ?

- Most foreseen changes are incremental
  - Switch from tool A to B
- ALICE plans major changes for LS2, basically a major rewrite: more commonality with off-line, all tools are under re-consideration. Working groups are starting now.
- Several experiments looking into GIT.

Use of Formal Software Development Processes

Use of Informal Software Development Processes

- Nobody admits to using any of the hot and important software processes that were in vogue about 10-15 years ago.
- Also nobody explicitly mentions any of the agile methodologies that are in vogue for the last 5-10 years.
- Mostly ad-hoc steps (write-up a list of requirements, draw some diagram to explain your solution etc.)
- CMS has a detailed document describing their procedure for getting changes into the DAQ software – mostly to protect the existing running system.
- Other experiments let the developer free hand and go through a release integration/testing/validation step.

Summary

- Wide agreement on basic tools, as expected (languages, compilers, operating systems, hardware)
  - People seem to converge on tools that are supported and “good enough” - e.g. Subversion even if it's not the latest and hottest thing.
- But convergence also in places where I personally didn't expect it (e.g. use of RPMs, handling of binary compatibility)
- The overall organization of every experiment specific software seems to lead to certain solutions, with its own set of constraints.
  - If you switch from one mode to another (ALICE?) seeing the experience of the other experiments might be very useful.

Future

- Is there a move to even more commonality, e.g. CMake ?
- In deploying user software via RPM, see e.g. RedHat Software Collections:
  - https://access.redhat.com/knowledge/docs/en-US/Red_Hat_Developer_Toolset/1/html/Software_Collections_Guide/
- It would be interesting to learn about the experience of others in pushing their current process to the next steps
  - E.g. going from nightly builds to continuous integration
- There are cross-experiment software meetings at CERN, mostly focused on frameworks, parallel processing etc.
  - Should there be the occasional online software meeting ?
- The fact that we are doing very similar things means that we can also easily profit from each other much more easily...