LHCb Perspective on the Future

M. Clemencic *on behalf of LHCb Computing*

June 7, 2016

CERN - LHCb

Table of contents

1. Software Management
2. Computing For LHCb Upgrade

Software Management

Migration to Git Rationale

- Many requests from developers
- Better than SVN
  - faster
  - distributed
  - better management of tags and branches
- Large community
  - newcomers may already know it
  - learning it is useful for students

Organization of Software Projects

LHCb software is organized in *projects* made of *packages*

- *package* $\equiv$ versioned entity
- *project* $\equiv$ releasable entity
- main applications built on top of common projects
- loose but stable coupling projects $\leftrightarrow$ packages
- migration from CVS to SVN tightened the coupling

Mapping SVN onto Git

A Git repository is a versioned unit.

Mapping SVN onto Git

A Git repository is a versioned unit.

Different approaches possible

- one repository per package
- one repository per project
- one repository for everything

all with pros and cons.

Mapping SVN onto Git

A Git repository is a versioned unit.

Different approaches possible

- one repository per package
- one repository per project
- one repository for everything

all with pros and cons.

We chose one repository per project.

Keeping the History

The command `git-svn` can migrate the whole history...

Keeping the History

The command `git-svn` can migrate the whole history…

… only for *proper* projects.

Keeping the History

The command `git-svn` can migrate the whole history…

… only for *proper* projects.

Our approach:

- create a Git repository from released versions of projects
- clean up SVN projects `trunk`s
- bind Git repositories to SVN `trunk`s (à la `git-svn`)
- keep Git repositories in sync with SVN
    - SVN commits pushed to Git master
    - GitLab merge requests automatically applied to SVN
- close SVN write access (when ready, project by project)

Developers Point of View

- Whole project development
    - simpler than with SVN
    - plain Git based approach
    - requires tools to set up build environment

Developers Point of View

- Whole project development
  - simpler than with SVN
  - plain Git based approach
  - requires tools to set up build environment
- Satellite projects
  - developed a few Git subcommands
  - keep customization to a minimum
  - non-standard use of Git can be unsettling

Developers Point of View

- Whole project development
    - simpler than with SVN
    - plain Git based approach
    - requires tools to set up build environment
- Satellite projects
    - developed a few Git subcommands
    - keep customization to a minimum
    - non-standard use of Git can be unsettling
- Software contributions
    - GitLab based workflow
    - feature branches → merge requests
    - code review (optional)
    - multiple production branches (devel, stable, ...)
        - not easy to keep track of where bug fixes went

Computing For LHCb Upgrade

Preparing the Upgrade

LHCb detector will be upgraded for Run 3

- 40MHz read-out
- software only trigger

Preparing the Upgrade

LHCb detector will be upgraded for Run 3

- 40MHz read-out
- software only trigger

Preparing the Computing TDR (due end 2017) on several fronts

- software framework
- event model
- non-event data
- hardware and data-flow
- data processing & analysis models
- simulation

Preparing the Upgrade

LHCb detector will be upgraded for Run 3

- 40MHz read-out
- software only trigger

Preparing the Computing TDR (due end 2017) on several fronts

- software framework
- event model
- non-event data
- hardware and data-flow
- data processing & analysis models
- simulation

Software Framework

Aiming for the migration to multithreaded Gaudi

- GaudiHive is a good starting point
  - control & data flow scheduler
  - backward compatibility
- we need something more
  - backward compatibility is a burden we cannot sustain
  - move to re-entrant stateless algorithms
  - direct configuration of control flow

Stateless Algorithms

Backward compatible multithreading cannot scale forever.

Stateless Algorithms

Backward compatible multithreading cannot scale forever.

Stateless algorithms mean

+ easier thread safety
+ better scalability
+ leaner code
- migrating a lot of code

Stateless Algorithms

Backward compatible multithreading cannot scale forever.

Stateless algorithms mean

+ easier thread safety
+ better scalability
+ leaner code
- migrating a lot of code

```
class MySum: public TransformAlgorithm<OutputData(const Input1&, const Input2&)> {
 MySum(const std::string& name, ISvcLocator* pSvc)
   : TransformAlgorithm( name, pSvc,
                         { KeyValue("Input1Loc", "Data1"),
                           KeyValue("Input2Loc", "Data2") },
                         KeyValue("OutputLoc", "Ouptut/Data") ) {
 }
 // ...
 OutputData operator()(const Input1& in1, const Input2& in2) const override {
   return in1 + in2;
 }
 // ...
};
```

Event Model

Old Event Model based on C++98 optimization

- pointers and arrays of pointers
- ownership not enforced
- inheritance
- memory fragmentation

New Event Model for new C++ and hardware

- structure of arrays
- inheritance free (type erasure)
- POD structures

Misc

Investigating in other contexts

- Non-Event Data
    - decoupling Detector Description and Conditions
    - DD4HEP and CMS/ATLAS CondDB project
- Hardware and Data-Flow
    - vectorization and GPUs
- Data Processing & Analysis Models
    - event tag collections
    - analysis trains
- Simulation
    - fast simulation

just few examples

Hackathon Outcomes

Core Software Hackathon on May 26-27

- control flow direct configuration
- integration of stateless algorithms and `DataHandle`s
- progress toward inheritance free event model

to be merged in Gaudi master

Summary

Summary

We’re not yet in the future, but we’re approaching it at a steady pace.

- switch to Git by the end of the year
- progressing on framework evolution
    - multithreading, vectorization, etc.
- milestones/checkpoints defined for Upgrade TDR
    - demonstrate feasibility by 2017 Q1
    - define migration plan by 2017 Q2