Configuration Management review status

LABORATOIRE DE L'ACCÉLÉRATEUR LINÉAIRE

Author: *Sebastien Binet*
Institute: *LAL/IN2P3*
Date: *2010-12-02*

Mandate

Review, collect feedback and then improve usage/performances of:

- Release building
- Release distribution
- Release managing
- Simple analysis running
- Single package development
- Power-user development
- Usage at CERN
- Usage outside CERN

Status / organization

- Current report is available on CDS with reference `ATL-COM-SOFT-2010-014`
  - `http://cdsweb.cern.ch/record/1266290`
- created work groups to work on each of the areas recognized during the review
- (not completely) up-to-date wiki page:

  `twiki:Atlas/CMR10WorkGroups`

- reports on the on-going work during bi-weekly SIT meetings

Environment setting - easy/uniform/fast

- DavidQ has been working on that with `asetup`
  `twiki:Atlas/AtlasSetup`
- latest versions of `asetup` pave the way towards a more integrated and uniform development environment
  - configuration of `tdaq`, `Gaudi` and `LCGCMT` projects
- in the works:
  - leverage new features of `CMT-v1r22` to speed-up the environment setup
  - building of a single per-project `requirements` file to setup a whole project (and its children) and tackle the `stat`-access of gazillions of `requirements` files
  - integrate/consolidate with already existing `AtlasXyzRuntime` packages

Speed of Building

- latest `CMT-v1r22` version tackles a few of the issues raised during the review
  - Ensure QUICK mode can be used for from scratch build
  - Introduce command `cmt build constituents_config` to generate constituents Makefile
  - generate/track source dependencies to minimize rebuild
- most important issue to tackle
  - reduce turnaround edit/compile cycle
  - paramount to have more analysis/reco/... contributions to `Athena`
- currently 2 avenues to address this problem
  - improve `CMT` itself (G. Rybkine)
  - investigate leveraging new tools to perform the build (while keeping `CMT` for the environment setup)
    - ★ test version of a `GAUDI``+``LCG` build using `CMake` (P. Mato)
    - ★ test version of `AtlasCore` build using `waf` (S. Binet)

Speed of building - II

- Pere's approach:
  - have a little `python` script executed for each `CMT` package
  - for each package, use `cmt` to get the list of `constituents`, compilation flags, include dirs, ...
  - from these gathered informations, build the `CMakeLists.txt` automatically
  - then execute the usual `CMake` commands to build
  - a few notes about `CMake`:
    - same approach to build than `CMT`: generates `Makefile` for each platform (windows, unices,...) and reuse the platform's build- and toolchain
- theoretically a smooth transition path
- my approach is similar
  - but it creates a `wscript` file, which is the `Makefile` equivalent for `waf`
  - `waf` is similar to `SCons` and `Cons`: a `python` library to steer the build and manage dependencies (so, no `Makefile`)

Speed of building - III

	cmt	cmake
GAUDI (noop)	50 s	7 s
GAUDI (full)	613 s	148 s
LHCB (noop)	480 s	17 s
LHCB (full)	2700 s	356 s
REC (noop)	335 s	13 s
REC (full)	1594 s	332 s

X ~4

X ~7

X ~5

see:

http://indico.cern.ch/getFile.py/access?contribId=2&resId=1&materialId=slides&confId=105778

---

http://www.cmake.org/

Speed of building - IV

- with `waf`
  - `http://code.google.com/p/waf/`
- caveats:
  - (re)started working on this just last week
  - not feature complete (`POOL` converters, `jobo` installation, ...)
  - only tested on `AtlasCore` packages
  - didn't test if build was fully functional (my favorite jobo worked)
- on a local install, with `AthenaPython`, `Valkyrie`, `AthenaBaseComps`, `AthenaKernel`, `SGComps`, `PerfMonComps`, `SGTools`, `AthenaServices`, `GaudiSequencer`, `PerfMonTests`, `PileUpComps`

Speed of building - V

- full first build:

```
$ cmt bro make -j8
480.88s user 1113.64s system 284% cpu 9:21.21 tota

$ waf configure clean build -j8
320.77s user  267.52s system 395% cpu 2:28.58 tota
```

- modifying `AthenaKernel/IThinningSvc.h`:

```
151.16s user 347.52s system 205% cpu 4:02.85 total
 24.47s user  15.70s system 325% cpu  12.329 total
```

- `touch AthenaKernel/IThinningSvc.h`

```
152.03s user 347.11s system 204% cpu 4:03.90 total
  1.00s user   0.35s system  82% cpu    1.65 total
```

Release build streamlining

- integration of `Gaudi` project into nightlies' builds
- still a few hiccups
  - different tagging conventions
  - we share the `Gaudi SVN` repository...
- reduce the length of various paths
  - `flat-slim.py` does this (creates a bunch of symlinks)
  - tests have been so far very limited
- next step would be to integrate `LCGCMT`
  - or at least parts of `LCGCMT`
    - `ROOT`, `POOL`, `COOL`, `CORAL`
  - `LCGCMT-externals` in a later stage

---

meanwhile, in `CMSSW`:

- build time: ~**6-7h** on a 10-core machine
  - binutils+gcc+ROOT+Geant4+...+pure-CMS-code+RPM-build
- build time: ~**2-3h** on a 10-core machine
  - pure-CMS-code

Conclusions

- first optimization results encouraging
- progress has been made on various fronts

- **but** we probably need somebody a pro-active shepherd to steer the various working groups
  - reports at SIT meetings have been somewhat sparse
    - ★ (I am certainly guilty of that too)