Status of HLT Integration
LS1 evolution - PSC and athenaHLT

Ricardo Abreu
Werner Wiedenmann
Frank Winklmeier

CERN

July 19, 2013
LS1 evolution of PSC and athenaHLT caused by evolution of TDAQ
- foundation is the merge of L2 and EF
- The origin of the changes is on the **HLT Interface**
  - the main **point of integration** with the HLT

**HLT Interface**
- Package that provides the API that the PUs and the HLT use to communicate
- Composed of mainly two parts:
  - FSM API → **used** by PU; **implemented** by HLT
  - DC API → **used** by HLT; **implemented** by PU
- Changes on the interface affect both sides
Evolution of the PSC and athenaHLT can be seen as deriving from changes in the HLT Interface

PSC framework
- The part that concretizes the HLT Interface on the HLT side
  - it wraps the rest of the HLT so that it can be executed online

athenaHLT (old athenaMT/PT)
- An alternative concretization of the online dataflow (PU+...)
  - allows the offline execution of the HLT without having to run a partition
  - useful for dev, auto testing, and reprocessing
LS1 changes on HLT Interface

Changes to the HLT Interface

1. One **single interface** (following L2 style for process) → *Merge*
2. Transitions receive **parameters in boost::property_trees** → *Ptrees*
3. New transitions for multi-process handling → *Forking*
4. **New data collection** scheme, with ROB reserving and monitoring → *DC*
Impact Item 1 – Merge

Summary

One **single interface** (following L2 style for process) → **Merge**

In general, it involves:

▷ preservation/adaptation of logic from one of the levels
▷ accumulation of logic from both levels
▷ renaming of things (from L2/EF to HLT)
   - the notions of L2/EF were fundamental and infused many code fragments
   - type names, includes, error ids, messages, comments, control flow constructs (like if-else)
   - Done, except where "external" software is involved
     - not evolved yet (e.g. eformat)
     - not integrated yet (e.g. trigger flags)
▷ used opportunity to improve and reorganize the code

Work simultaneous with ptrees

▷ Items 3 and 4 clearly separated → can be done separately from the rest;
▷ Work for items 1 and 2 (ptrees) is naturally coupled → both change same element of HLT Interface
Impact Item 2 – Ptrees

Summary

Transitions receive **parameters in** `boost::property_trees → Ptrees`

- Allows arbitrary communication from DF to HLT
  - No need to change HLT Interface when information exchange changes
- The HLT Interface abstains from specifying the communication contents
- Of course, both sides still need to agree
  - Detection of problems passed to run-time when the compiler could have been done it → accepted downside

In practice...

- Approach replaces 2 ways of receiving information in the HLT: direct parameters and global info (e.g. OKS, RunParams)
- The only transitions where *any* information is being inserted in the ptrees are `configure` and `prepareForRun`
  - In the future, also `prepareWorker` and possibly `finishWorker` and `hltUserCommand`
  - Biggest chunk of the work on `configure`
- Work on this point consisted mainly on:
  - Establishing contents/structure of the ptrees and aligning with other developers
  - Fill them in according to the user’s intentions
  - Read and use them, replacing previous approaches
New transitions for multi-process handling → *Forking*

- In order to profit from the CoW feature, the HLTPU forks into several processes during PU’s `prepareForRun`
- after HLT’s `prepareForRun`

- the new transitions are: `prepareWorker` and `finishWorker`

```
HLTPU::prepareForRun()
{
  ...
  // the HLT does its prepareForRun here
  HLTInterface.prepareForRun(...)
  
  // the HLT finished preparing. Now we fork
  fork_result = fork();
  
  if (fork_result == IN_CHILD_PROCESS)
  {
    // here we know we are in one of the forked units
    // we need to setup individual stuff
    HLTInterface.prepareWorker(...)
  }
  else // IN_MOTHER_PROCESS
    setupWhateverIsNeededForMother();
}
```
But why are new transitions needed?

▷ Some of the parameters that are set in the configuration are meant to be unique to each process (e.g. PID)
▷ after the forking, each process has copies of these parameters → a new transition is needed to individualize them (prepareWorker)

Potential pitfall

▷ By the time the prepareWorker transition is reached, copies of the original info may already have been created by the software down the stack.
▷ To detect these copies:
  1. make info unavailable before prepareWorker
  2. see where problems appear (i.e. places where the info is expected)
  3. solve by moving code to prepareWorker or by overwriting existing copies
▷ In practice, not many cases expected

▷ When coming back from running, forked processes need a chance to cleanup
  finishWorker called during HLTPu’s stopRun but after HLT’s stopRun
New data collection scheme, with ROB reserving and monitoring → DC

In the old days...
- L2 received a L1R and requested necessary ROBs
  - requests packed and cached by the ROBDataProviderSvc
- EF received already the full event

In the new scheme...
- The HLT follows the L2 approach for the DC, but
  - caching and packing managed entirely by the Data Collection Manager (DCM)
  - ROBDataProviderSvc only forwards requests to reserve and collect data to the DCM
- DC component of the HLT Interface changed to:
  - include a reserve method
  - piggyback cost-monitoring relevant data on the collect method (e.g. was the ROB cached)
- ROBDataProviderSvc interface to HLT algorithms stays the same

athenaHLT has to implement the Data Collection
- reproducing the behavior of the DCM
- some aspects remain to be defined (e.g. from what threshold of cached ROBs will the DCM get the full event)
Global progress

Changes to the HLT Interface

1. One single interface (following L2 style for process) → Merge
   - only sparse changes left, to integrate external code

2. Transitions receive parameters in boost::property_trees → Ptrees
   - RunParams still read from IS
     - some discussion still ongoing regarding future set of run params
   - HLTImplementationDB config from athenaHLT also pending (--use-database)
     - some reflection still needed (see twiki for details)

3. New transitions for multi-process handling → Forking
   - To do...

4. New data collection scheme, with ROB reserving and monitoring → DC
   - Werner’s work needs to be integrated
     - His simulation of DCM’s job has to be passed to athenaHLT

5. Much necessary code restructuring/rewrite

Focusing on athenaHLT

Examples of things that have been done...

### Python bindings for the new HLT Interface

- including ptree bindings that try to incorporate python spirit
  - `myptree['path.to.my.stuff'] = 'foobar'`
  - same approach followed for iterators, key/value concepts, membership tests, etc.

### New configuration strategy

- further encapsulate option specification and separate it from parsing
- implement a configuration class, following the RAII principle, which encapsulates the configuration
  - responsible for parsing and digesting the cli args
  - works with different option specifications (i.e. `file_based`, `emon` when supported)
  - provides a standard way to access configuration for the rest of the program
  - generates ptrees on the fly
  ```python
c = configuration(file_opt_spec, cli_args)
print "joboptions:", c['joboptions']
print "precommands:", c['precommand']
print "config ptree:", c.get_config_ptree()
```
Integrate processor with new configuration scheme

- Processor now receives configuration object in the constructor
  - (plus online infrastructure for future runs with online monitoring)
- old way of accessing attributes maintained but redirected
  - `processor.libraries` internally translated to `processor.config['libraries']`
  - `processor.run_number` internally translated to `processor.config['run-number']`
  - old code accessing the attributes could be maintained

Automatic testing and development scaffolding

- many unit tests
- some integration (application-level) tests (more to be added in time)
- In both cases
  - brought old tests up to speed / replaced outdated ones
  - added new ones
  - improved testing framework
  - made test failure detectable with exit code
- Include notions of unsupported and untested options
  - Using an option with a value that wasn’t explicitly marked as supported results in an error
  - Using an option with a value that was marked as untested results in a warning
Many features that were supported in athenaMT/PT are already supported in athenaHLT

Most still marked as untested

Examples

- input event modifiers
- saving output events with different compression levels
- interactive more with python prompt
- static and dynamic run-number overwriting
- timeout watchdog
- Changing the PSC’s setup python file
- Pre and post commands
- etc.
Practicalities

- athenaHLT is already available in the SVN (along with the PSC)
  - Tags TrigKernel-20-00-00, TrigServices-20-00-00, TrigPSC-20-00-00, and HLTTestApps-20-00-00
- However, it requires the new HLT Interface
  - which requires C++11
- Have currently been working with a custom HLT Interface
- Should go into a nightly soon - dev/devval or MIG8?
  - compile against tdaq-05-00-01 / C++11 offline nightly?
- Move of hltinterface package to latest tdaq-common (01-23-00)
  - Will be used in dev/devval in the next few days, leading towards release 18
  - PSC packages will have no more dependency on tdaq and can be moved to AtlasTrigger later on