Configuring LHCb’s High Level Trigger

Rosen Matev, Niklas Nolte, Alex Pearce on behalf of the LHCb collaboration

CHEP 2019 - Track 5
Data processing at LHCb

➢ Hadron collider → lots of data
➢ Trigger system selects a tiny, interesting fraction
➢ What is interesting?
  ➢ Depends on your goals
➢ Year 2021+:
  ➢ more versatility please → no more hardware trigger stage
  ➢ need to be fast!
  ➢ need to be correct!
  ➢ many physicists → many different goals → many different signatures to trigger on → a huge collection of selections (and people) to deal with
Data processing at LHCb

- Codebase in C++
- Algorithms implemented in a functional way:
  - Output operator()(Input const&...) const // ← pure function
- Assembling of algorithm instances controlled via python
Assembling what exactly?

- Data Flow
- Configurable algorithm properties
- Inputs & Outputs
- Control Flow
  - What to run and when to stop

This does not only concern the trigger!
Trigger line anatomy

Data flow

Control flow

hits
muon PIDs
tracks
fitted tracks
primary vertices

Prescale?

Pass global event cut?

Sel(PT > 400 & P > 5000)
Sel(IsMuon)
PreSelected Muons

Any PreSelectedMuons?

Sel(CHI2DOF < 2.5 & FakeProb < 0.2)
Sel(MVA(PT, IP) > 1.1)

Any SelectedMuons? (decision)

Selected Muons
Python representations of Algorithms (Configurables) implemented as Singletons:

```python
>>> from Configurables import Tracking
>>> Tracking("test") is Tracking("test")
True
>>> Tracking() is Tracking()
True
```

Some functions define big chunks of logic and spit out explicit sequences

- “setupHlt1Tracking”
- Many parameters to control the behaviour
Data flow configuration: Go functional

- Small functions with a clear objective
- Goals:
  - No globally mutable instances!
  - Instantiated Algorithms → Immutable
  - “Obvious” data flow
  - deduce dependencies behind the scenes
  - Only minimal control flow, hard to mess up
Example

```python
@configurable
def make_long_tracks(make_tracks = make_upstream_tracks, make_hits = make_scifi_hits, minimum_PT = 400*MeV):
    
    """Makes long tracks from upstream tracks and scifi hits""
    
    # prepare inputs
    hits = make_hits()
    upstream_tracks = make_tracks()

    # setup tracking algorithm
    tracking = Tracking(Hits = hits,
                         InputTracks = upstream_tracks,
                         min_PT = minimum_PT)

    # give back the long tracks
    return tracking.OutputTracks
```
Visualization

make_velo_tracks().plot()
@configurable

def make_long_tracks(make_tracks = make_upstream_tracks,
                      make_hits = make_scifi_hits,
                      minimum_PT = 400*MeV):
    """Makes long tracks from upstream tracks and scifi hits""
    # prepare inputs
    hits = make_hits()
    upstream_tracks = make_tracks()

    # setup tracking algorithm
    tracking = Tracking(Hits = hits,
                         InputTracks = upstream_tracks,
                         min_PT = minimum_PT)

    # give back the long tracks
    return tracking.OutputTracks
Key feature: bind

- Users need to control the application from the top level:
  - Run only parts of the reconstruction
  - Change a cut
  - Replace an algorithm
  - ....

- “bind” : capable of binding new values to arguments of functions
Key feature: bind

```python
@configurable
def filter_tracks_by_P(min_P = 1*GeV):
    tracks = Filter(min_P)
    return tracks

def use_tracks():
    tracks = filter_tracks_by_P()
    do_something(tracks)

with filter_tracks_by_P.bind(min_P = 2*GeV):
    use_tracks()
```
Defines a line

Select Kaons & Pions

Combine them to D⁰ mesons

Control flow: Short circuited AND
If one selects 0 particles
→ we stop right there
C++ Algorithm implementations are put into context with python

- Functional approach to data flow configuration
  - “deep” configurability with `bind`
  - No mutable state
  - Run any data producer → (visual) information of the relevant dataflow
  - Dependencies deduced, no explicit sequences

- Control flow constructed in a tree-like graph
  - Leaves: trigger lines
  - C++ event scheduler traverses the graph as constructed in python
Conclusion & Outlook

- Some features are missing, quite some work still ongoing
- quite different with respect to Run 1&2 configuration
  → some confusion occurred

But:
People have already successfully written…

- Trigger lines
- Reconstruction configuration

We have received feedback such as “Writing selections is quite intuitive”
Thanks
Visualization of control flow
Data processing at LHCb

➢ Codebase in C++

➢ Algorithms implemented in a functional way:
  ➢ Output operator()(Input const&...) const // ← pure
  ➢ Get immutable Input from EventStore (by key)
  ➢ Transform data without side effects
  ➢ Push Output to EventStore (with key)

➢ Assembling of many many algorithm instances controlled via python
Python representations of Algorithms (Configurables) implemented as Singletons:

- **Pro:**
  - Easy access by name
  - Change of properties from anywhere!

- **Con:**
  - Change of properties from anywhere! → hard to track changes
  - You can overwrite properties without realizing

```python
>>> from Configurables import Tracking
>>> Tracking("test") is Tracking("test")
True
>>> Tracking() is Tracking() # works even without explicit names
True
```
A few functions manage the assembly of algorithms in explicit sequences

Pro:
- Large bunches of logic can be found in one place

Con:
- Little granularity:
  - Easiest way to run the Velo reconstruction?
  - First set up the full track reconstruction
  - Remove everything but the Velo part