Of Frames and schema evolution - The newest features of podio

P. Fernandez Declara, F. Gaede, G. Ganis, B. Hegner, C. Helsens, T. Madlener, A. Sailer, G. A. Stewart, V. Volkl

Introduction

- podio is an event data model (EDM) toolkit
- Generate efficient and thread safe c++ code from a high level description in YAML
- Favor composition over inheritance and use plain-old-data (POD) types where possible
- Layered design allows for efficient memory layout and performant I/O implementation
- User Layer consists of thin handles that offer *value semantics*
- Support multiple I/O backends
  - ROOT (default), SIO

```cpp
auto particles = MCParticleCollection();
// ... fill ...

for (auto mc : particles) {
  auto mass = mc.getMass();
  for (auto p : mc.getParents()) {
    auto pid = p.getPDG();
  }
}
```

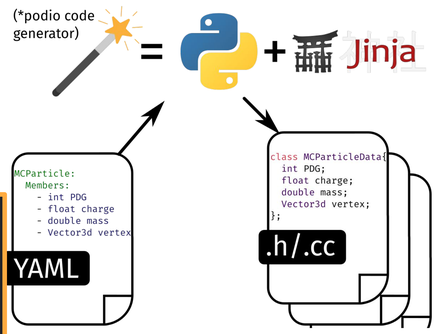
C++17 code with *value semantics*

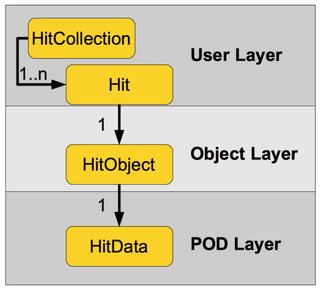
```python
particles = MCParticleCollection()
# ... fill ...
for mc in particles:
  mass = mc.getMass()
  for p in mc.getParents():
    pid = p.getPDG()
```

python bindings via *PyROOT*

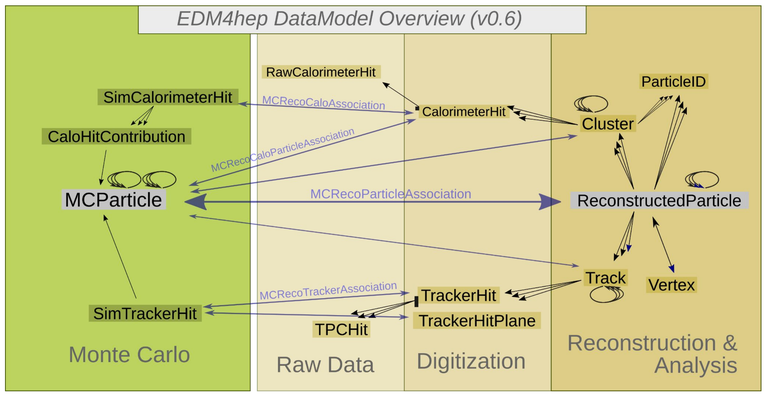
```python
d = ROOT.RDataFrame('events', 'events.root')
h = (d.Define('abs_pdg', 'abs(Particle.PDG)')
      .Define('mu_sel', 'abs_pdg == 13')
      .Define('mu_px',
              'Particle.momentum.x[mu_sel]')
      .Histo1D('mu_px'))
h.DrawCopy()
```

Easy to read ROOT files





podio generated EDMs



Recent developments

- Possibility to extend existing datamodel defintions
  - Used by EIC (extending EDM4hep)
  - Allows for prototyping of new datatypes
- Conversion to JSON output using `nlohmann/json`

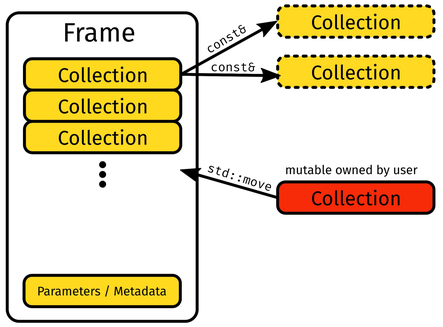
New

This project has received funding from the European Union's Horizon 2020 Research and Innovation programme under grant agreement No 101004761.

`podio::Frame`

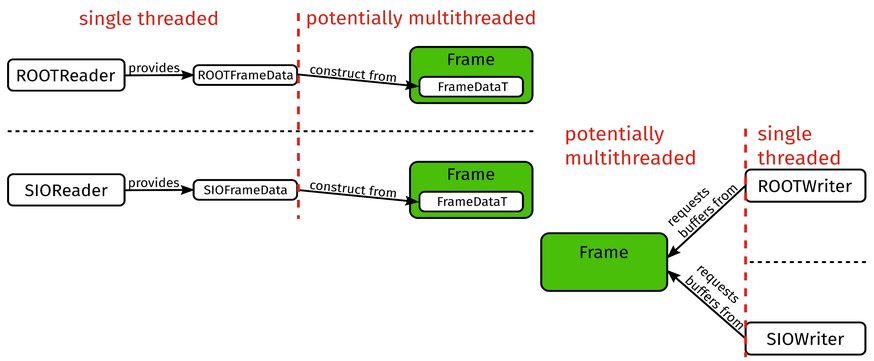
Main Ideas

- Container aggregating all relevant data
- Defines an *interval of validity* or category for the contained data
  - Event, Run, readout frame, ...
- Offers easy to use and thread safe interface to access data
  - Immutable read access only
  - Enforced move for insertion



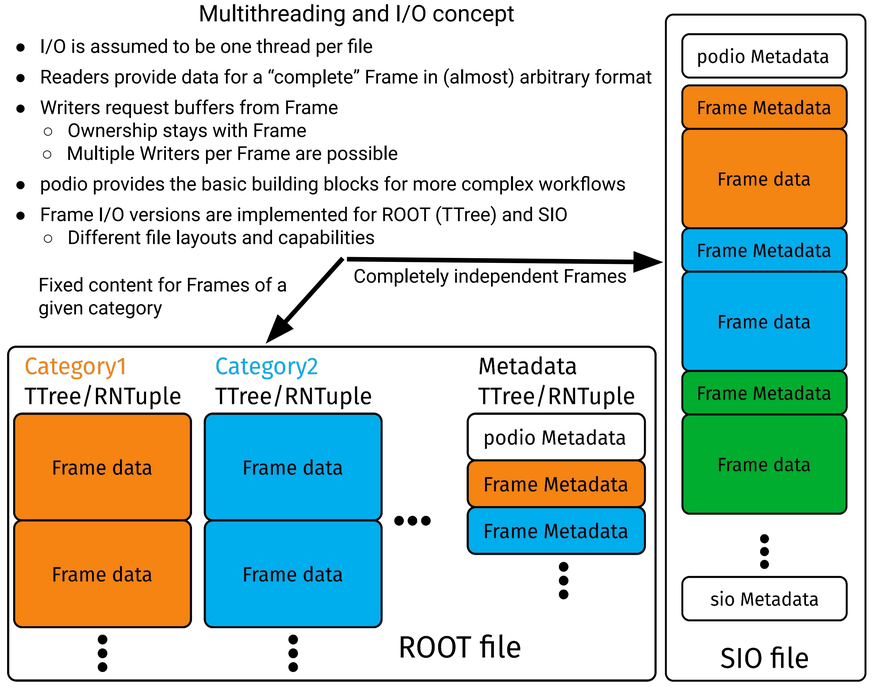
Some Implementation Details

- The `podio::Frame` is a move-only type
- *Type erasure* for value semantics and templated get/put functionality
  - Allows constructing from "arbitrary" data
  - Hides policies that affect behavior transparently



Multithreading and I/O concept

- I/O is assumed to be one thread per file
- Readers provide data for a "complete" Frame in (almost) arbitrary format
- Writers request buffers from Frame
  - Ownership stays with Frame
  - Multiple Writers per Frame are possible
- podio provides the basic building blocks for more complex workflows
- Frame I/O versions are implemented for ROOT (TTree) and SIO
  - Different file layouts and capabilities



Schema evolution

To allow for long-term storage PODIO provides the necessary functionality for schema-evolution.

The description of data models in YAML files allows for an automated analysis of changes.

```yaml
schemaversion: 1
<...>

ExampleStruct:
  Members:
    - int x
    - double y

<...>
```

```yaml
schemaversion: 2
<...>

ExampleStruct:
  Members:
    - int x
    - int y

<...>
```

```
Comparing datamodel versions v2 and v1

Found 4 schema changes:
 - 'ToBeDroppedStruct' has been dropped
 - 'ex2::NamespaceStruct' has an addded member 'y'
 - 'ex2::NamespaceStruct' has a dropped member 'y_old'
 - 'ExampleStruct.x' changed type from int to double

Warnings:
 - Definition 'ex2::NamespaceStruct' has a potential member rename
'y_old' -> 'y' of type 'int'.

ERRORS:
 - Forbidden schema change in 'ex2::NamespaceStruct' for 'x' from
'std::array<int, 2>' to 'int'
```

Non-trivial changes are reported to the user as errors. Depending on user demands, those may get supported in the future. As the schema evolution happens on data read, old files can still benefit from features added later.

Backend Support

For the ROOT backend, PODIO does a sanity check, whether the change is supported by its automatic schema evolution and forwards the transformation work to ROOT itself.

For other backends like SIO, PODIO auto-generates transformation code, which is being called during the `Frame` read