EDM4hep.jl: Analyzing EDM4hep files with Julia

Pere Mato, CERN (pere.mato@cern.ch)

Abstract

EDM4hep[1] aims to establish a standard event data model for the store and exchange of event data in future HEP experiments, thereby fostering collaboration across various experiments and analysis frameworks. The Julia package EDM4hep.jl can generate Julia-friendly structures for the EDM4hep data model and reading event data files in ROOT format (either TTree or RNTuple) that are written by C++ programs, utilising the UnROOT.jl package [2].

Introduction

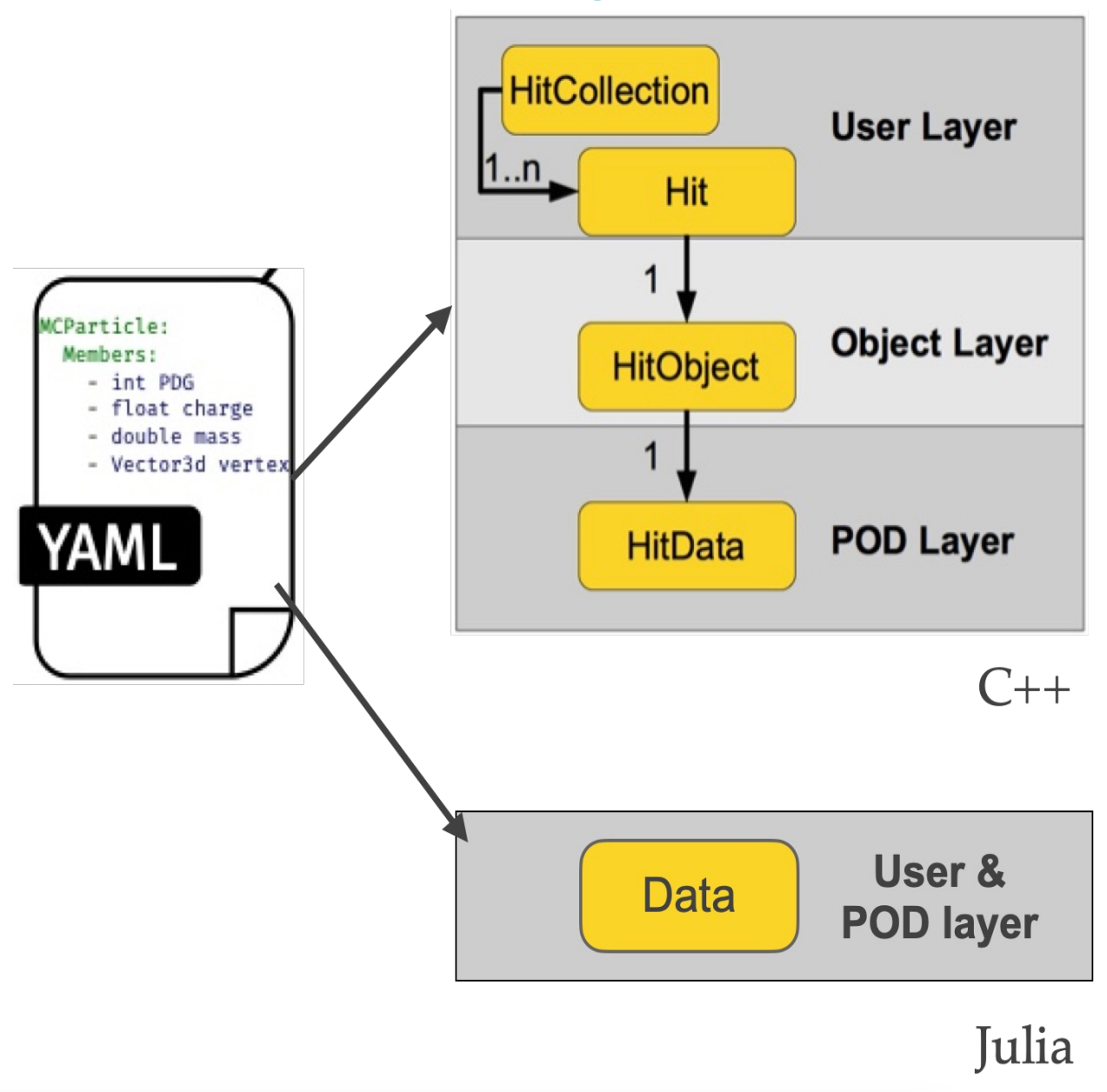
EDM4Hep is based on the PODIO edm-toolkit [3]. It uses yaml-files to define Event Data Model (EDM) data structures covering the simulation, digitalization, reconstruction and analysis domains.

A set of Python/Jinja scripts generate C++ code in three layers:

- POD layer - the actual data in array of structs
- Object layer - add relations and vector members
- User layer - thin handles and collections

For Julia, a single layer is generated with immutable 'user-friendly' structures

The default I/O backend is ROOT (TTree/RNtuple)



```
"""
struct MCParticle
    Description: The Monte Carlo particle - based on the lcio::MCParticle.
"""
struct MCParticle <: POD
    index::ObjectID{MCParticle}      # ObjectID of itself
   #---Data Members
    PDG::Int32                       # PDG code of the particle
    generatorStatus::Int32           # status of the particle as defined by the ...
    simulatorStatus::Int32           # status of the particle from the simulation ...
    charge::Float32                  # particle charge
    time::Float32                    # creation time of the particle in [ns] wrt. ...
    mass::Float64                    # mass of the particle in [GeV]
    vertex::Vector3d                 # production vertex of the particle in [mm].
    endpoint::Vector3d               # endpoint of the particle in [mm]
    momentum::Vector3f               # particle 3-momentum at the production vertex..
    momentumAtEndpoint::Vector3f     # particle 3-momentum at the endpoint in [GeV]
    spin::Vector3f                   # spin (helicity) vector of the particle.
    colorFlow::Vector2i              # color flow as defined by the generator

    #---OneToManyRelations
    parents::Relation{MCParticle,1}   # The parents of this particle.
    daughters::Relation{MCParticle,2} # The daughters this particle.
end
```

```
p1 = MCParticle(PDG=2212, mass=0.938, momentum=(0.0, 0.0, 7000.0), generatorStatus=3)
p2 = MCParticle(PDG=2212, mass=0.938, momentum=(0.0, 0.0, -7000.0), generatorStatus=3)

p3 = MCParticle(PDG=1, mass=0.0, momentum=(0.750, -1.569, 32.191), generatorStatus=3)
p3, p1 = add_parent(p3, p1)
...
```

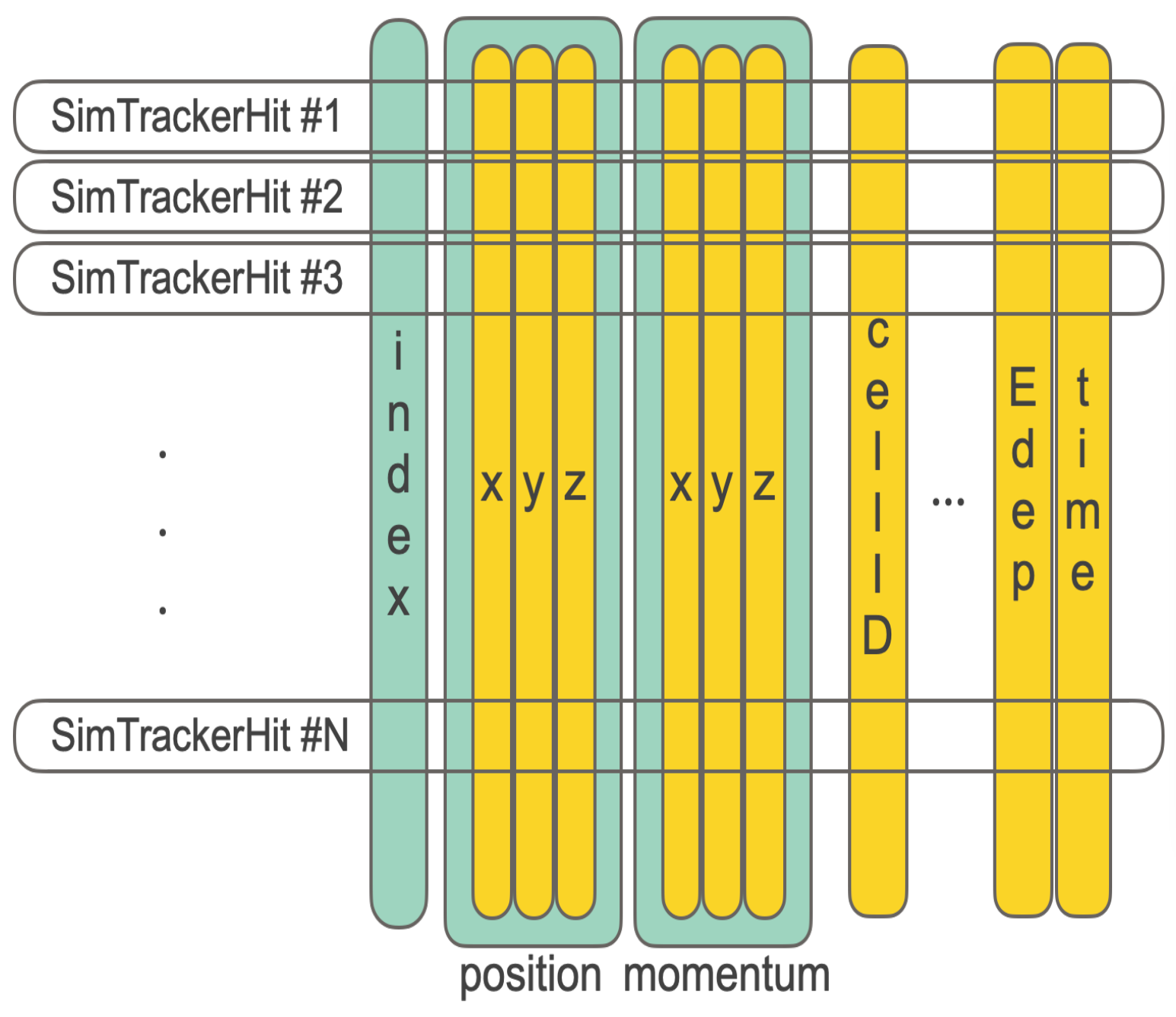
Reading EDM4hep ROOT files

Reading EDM4hep files is done using the UnROOT.jl package. It supports (transparently) TTree and RNTuple formats and several versions of PODIO.

Data files consist exclusively of 'collections-of-datatypes' (e.g. ReconstructedParticles, Vertices, etc.) identified by a 'collection-name'

The goal is to obtain a StructArray{DataType} of each collection for each event. SoA storage model in memory. Very efficient for columnar operations.

The exercise consists in mapping the schema in the ROOT file to the actual Julia datatype (using the Julia introspection and/or generated code)



```
julia> mcps = <get all MCParticle collection>

julia> typeof(mcps[1])      # get the first element
MCParticle

julia> length(mcps.charge) # mcps.charge is a Vector{}
211

julia> mcps[1:2].momentum  # slicing
2-element StructArray(…) with eltype Vector3f:
 (0.5000167,0.0,50.0)
 (0.5000167,0.0,-50.0)

julia> sum(mcps[1:2].momentum) # columnar operations
(1.0000334,0.0,0.0)
```

Objects are 'materialised' when requested (usually on the stack) to be able to call user object methods accepting these type as arguments (multiple-dispatch)

Reading Interface

Provided a simple interface for reading data files:

- EDM4hep files can be local or remote (e.g. root://eospublic.cern.ch/...)
- Single or multiple files
- Sequential and multi-threaded access

```
using EDM4hep
using EDM4hep.RootIO

reader = RootIO.Reader("ttbar_edm4hep_digi.root")
events = RootIO.get(reader, "events")

evt = events[1];

hits = RootIO.get(reader, evt, "InnerTrackerBarrelCollection")
mcps = RootIO.get(reader, evt, "MCParticle")

for hit in hits
    println("Hit $(hit.index) is related to MCParticle $(hit.mcparticle.index)
    with name $(hit.mcparticle.name)")
end

#---Loop over events-------------------------------------------------------------
for (n,e) in enumerate(events)
    ps =  RootIO.get(reader, e, "MCParticle")
    println("Event #$(n) has $(length(ps)) MCParticles with a charge sum of
$(sum(ps.charge))")
end
```

```
Hit #1 is related to MCParticle #65 with name pi+
Hit #2 is related to MCParticle #65 with name pi+
Hit #3 is related to MCParticle #65 with name pi+
Hit #4 is related to MCParticle #65 with name pi+
Hit #5 is related to MCParticle #66 with name pi-
Hit #6 is related to MCParticle #66 with name pi-
Hit #7 is related to MCParticle #66 with name pi-
Hit #8 is related to MCParticle #49 with name pi+
Hit #9 is related to MCParticle #49 with name pi+
...
```

~ 1500 times faster than Python interface

Multi-threaded Analysis

Developed mini-framework to ensure thread safety

- The user defines a data structure and an analysis function
- Each thread works on a subset of events using its own copy of the output data
- At the end, the results are 'summed' automatically

```
mutable struct MyData <: AbstractAnalysisData
    df::DataFrame
    pevts::Int64
    sevts::Int64
    MyData() = new(DataFrame(...), 0, 0)
end
```

```
function myanalysis!(data::MyData, reader, events)
  for evt in events
    data.pevts += 1                              #
    μIDs = RootIO.get(reader, evt, "Muon_objIdx")#
    length(μIDs) < 2 && continue                 # skip if less than 2

    recps = RootIO.get(reader, evt, "ReconstructedParticles")
    muons = recps[μIDs]                          # use the ids to subset

    sel_muons = filter(x -> pₜ(x) > 10GeV, muons) # select the Pt of muons
    zed_leptonic = resonanceBuilder(91GeV, sel_muons)
    zed_leptonic_recoil = recoilBuilder(240GeV, zed_leptonic)
    if length(zed_leptonic) == 1                 # filter exactly one Z
      Zcand_m        = zed_leptonic[1].mass
      Zcand_recoil_m = zed_leptonic_recoil[1].mass
      Zcand_q        = zed_leptonic[1].charge
      if 80GeV <= Zcand_m <= 100GeV              # select on mass Z
         push!(data.df, (Zcand_m, Zcand_recoil_m, Zcand_q))
         data.sevts += 1                         # count selected events
      end
    end
  end
  return data
end
```

```
events = RootIO.get(reader, "events")
mydata = MyData()
do_analysis!(mydata, myanalysis!, reader, events;  mt=true)
# mydata holds the summed results of the data analysis
```

If not all the attributes are needed, the user can define customized getters with a subset of attributes to optimize reading

```
get_μIDs  = RootIO.create_getter(reader, "Muon_objIdx")
get_recps = RootIO.create_getter(reader, "ReconstructedParticles";
                selection=[:energy,:momentum,:charge,:mass])

function myanalysis!(data::MyData, reader, events)
  for evt in events
    μIDs = get_μIDs(evt)
    length(μIDs) < 2 && continue # skip if less than 2

    recps = get_recps(evt)
    muons = recps[μIDs] # use the ids to subset the reco particles
  ...
  end
end
```

Results

- Sequential performance is pretty good compared to FCCAnalyses framework (Python+C++) with the higgs/mH-recoil/mumu example
  - ~21000 events/s compared with ~9500 events/s
- MT scalability is not great
  - Performance peak is reached with 8 cores (probably due to the garbage collector adding serial execution)

Conclusions

- The Julia package EDM4hep.jl[4] is registered in the Julia general registry and ready for use!

```
julia -e 'import Pkg; Pkg.add("EDM4hep")'
```

- Demonstrated how data analysis can be streamlined using high-level objects, offering a more intuitive and structured approach compared to flat n-tuples, all within a single, consistent and fast programming language.

[1] EDM4hep - https://github.com/key4hep/EDM4hep
[2] UnROOT.jl - https://github.com/JuliaHEP/UnROOT.jl
[3] PODIO - https://github.com/AIDASoft/podio
[4] EDM4hep.jl - https://github.com/peremato/EDM4hep.jl