Status of EDM4hep

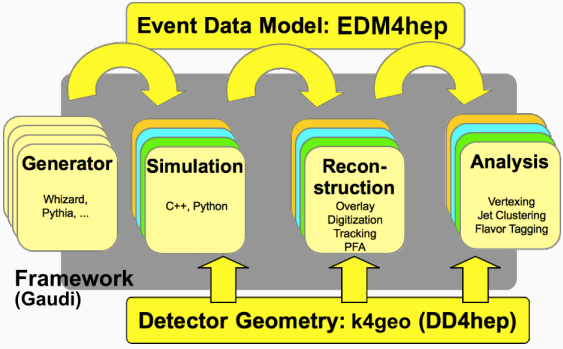
Thomas Madlener

7th FCC physics workshop

Jan 30, 2024

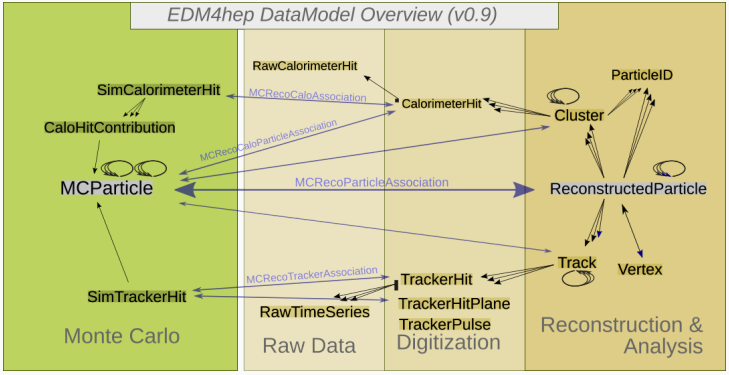
This project has received funding from the European Union's Horizon 2020 Research and Innovation programme under grant agreement No 101004761.

The EDM at the core of HEP software



- Different components of experiment software have to talk to each other
- The event data model defines the language for this communication
- Users express their ideas in the same language

EDM4hep - The common EDM for Key4hep



key4hep/EDM4hep

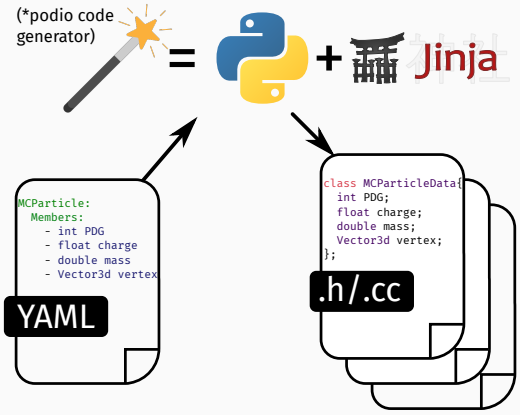
edm4hep.web.cern.ch

AIDASoft/podio

- Based on `LCIO` and `FCC-edm`
  - Focus on usability in analysis
- Quite stable over the last two years
  - **Some breaking changes foreseen for v1.0!**
- Can easily be extended
  - Used by EDM4eic
  - Prototyping!
- Generated via `podio`

The podio EDM toolkit

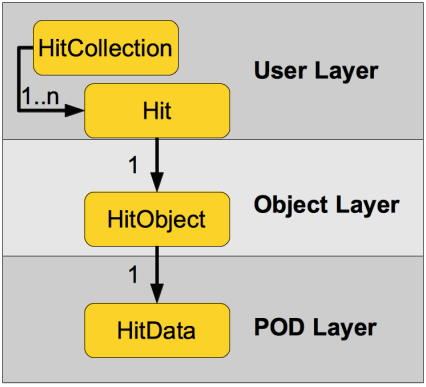
- Implementing a performant event data model (EDM) is non-trivial
- Use `podio` to generate code starting from a high level description
- Provide an easy to use interface to the users
- Main customers and feature drivers
  - key4hep/EDM4hep
  - eic/EDM4eic



AIDASoft/podio

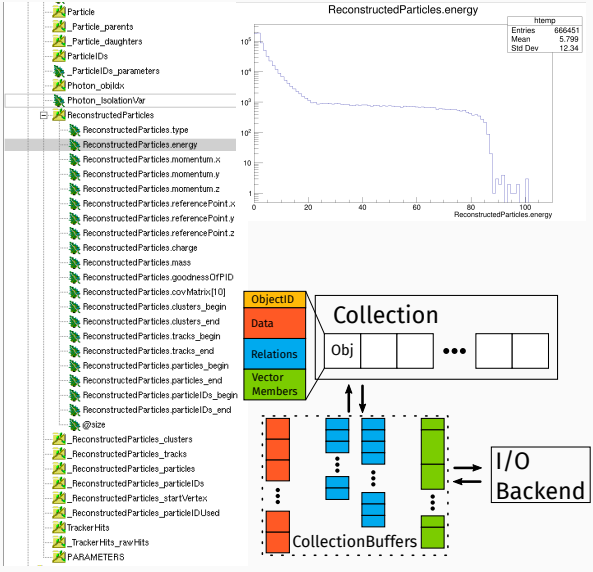
The three layers of podio

- podio favors **composition over inheritance** and uses **plain-old-data (POD)** types wherever possible
- Layered design allows for efficient memory layout and performant I/O implementation



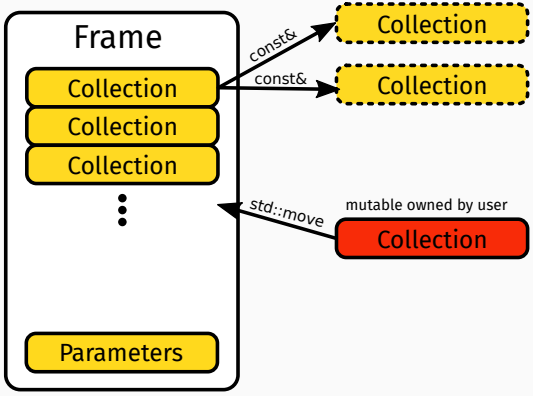
podio supports different I/O backends

- Default **ROOT** backend
  - Effectively flat ntuples (`TTree` / `RNTuple`)
  - Files can be interpreted **without EDM library**(!)
  - **Intelligble names** NEW Jun 23
  - Can be used in `RDataFrame` (**FCCAnalyses**) or with `uproot`
- Adding more I/O backends is possible
  - Alternative SIO backend exists
- **Many features only available through generated interfaces**



The `Frame` - A generalized (event) data container

- *Type erased* container aggregating all relevant data
- Defines an *interval of validity* / category for contained data
  - Event, Run, readout frame, ...
- Easy to use and thread safe interface for data access
  - Immutable read access only
  - Ownership model reflected in API
- Decouples I/O from operating on the data
- Old `EventStore` has been removed!



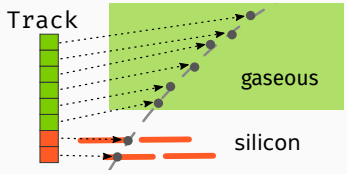

```
template<typename CollT>
const CollT& get(const std::string& name) const;

template<typename CollT, /*enable_if*/>
const CollT& put(CollT&& collection,
                 const std::string& name);
```

Schema evolution


```
Comparing datamodel versions v2 and v1

Found 3 schema changes:
 - '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 [...]

ERRORS:
 - Forbidden schema change in 'ExampleStruct' for 'x' [...]
```

- Allow to read old versions of an EDM from file and convert “on-the-fly”
- Hard problem with many considerations
  - Leverage backend if possible
  - Allow user defined evolution
- **Evolution always directly to current version**
- Detect potential problems at code generation
  - Expand available automatic evolutions as necessary
- **Machinery in place**; “whatever ROOT can do” for now

Interface types and their use in EDM4hep



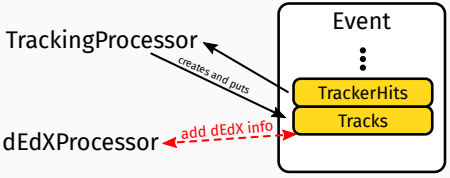

```yaml
interfaces:
  edm4hep::TrackerHit:
    Types: [edm4hep::TrackerHit3D, edm4hep::TrackerHitPlane]
    Members:
      - edm4hep::Vector3f position [mm] // hit position

datatypes:
  edm4hep::Track:
    OneToManyRelations:
      - edm4hep::TrackerHit trackerHits // hits of this track
```

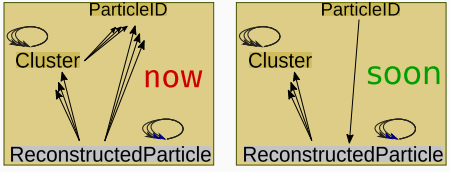
```cpp
auto track = edm4hep::Track{};
track.addHit(edm4hep::TrackerHit3D{});
track.addHit(edm4hep::TrackerHitPlane{});

const auto hits = track.getHits();
hits[0].isA<edm4hep::TrackerHit3D>();       // <-- true
hits[0].getValue<edm4hep::TrackerHit3D>(); // <-- "cast back"
hits[1].isA<edm4hep::TrackerHit3D>();       // <-- false
hits[1].getValue<edm4hep::TrackerHit3D>(); // <-- exception!
```

- General interface can be useful to "gloss over some details"
- Value semantics prevent inheritance based approach
  - Pointers in interfaces break consistency
  - No base class to inherit from
- Introduce *interfaces* as new category in YAML definition
  - Define desired functionality
  - **No collections!**
  - Use like normal *datatypes*
  - "Casting back" is possible

Current developments towards EDM4hep v1.0

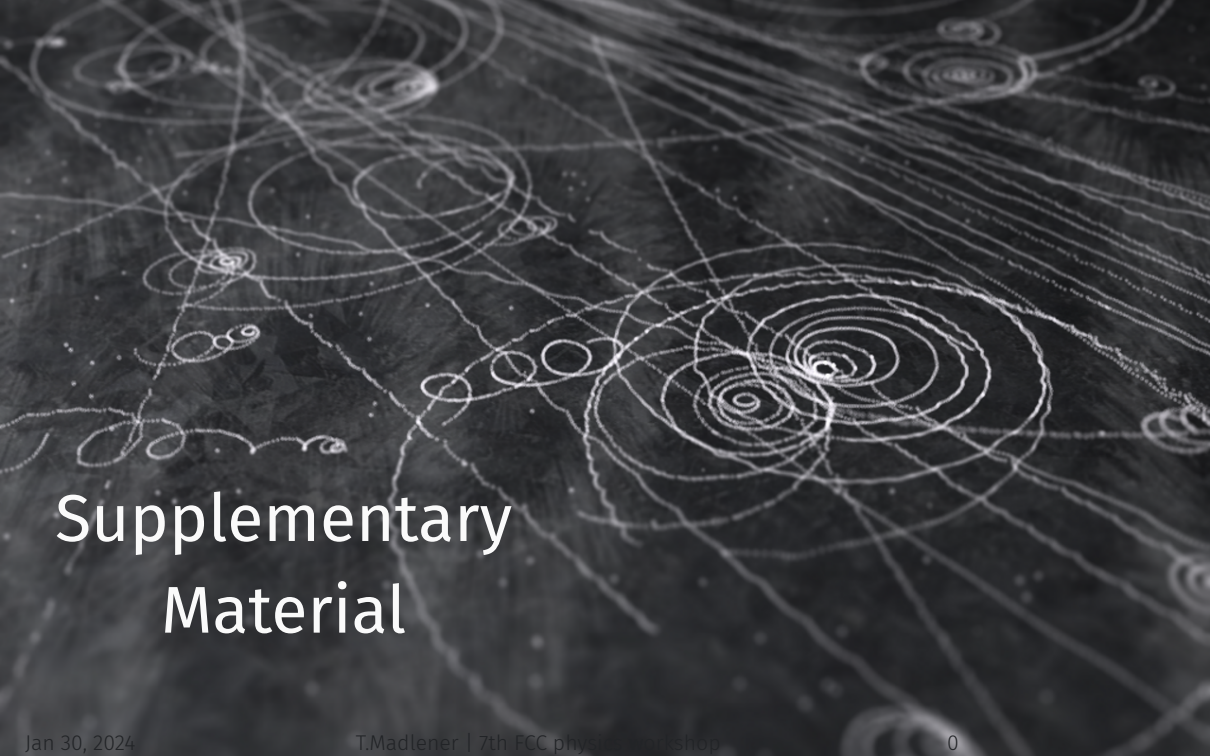
- **We are going to break some things without intention of doing schema evolution!**
- Consistent mutability concept
  - Some inconsistencies inherited from LCIO
  - Make sure to have relations “in the right direction”
  - Remove unused relations
- Introduce `TrackerHit` interface
- Add multiple weights to `EventHeader`
- Renaming of a few relations
- **Now is a good time to bring up bigger changes**
- Keeping track in EDM4hep v1.0 project





Summary

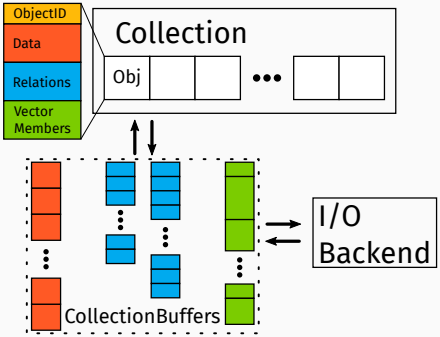
- EDM4hep is the common EDM and as such a core component of Key4hep
- Crucial developments in podio are done
  - File format frozen for quite some time now
  - Consider it *feature complete* for now
  - v1.0 soon (weeks) $\rightarrow$ backward compatible from then on
- Need to fix some conceptual issues and integrate podio latest features in EDM4hep
  - Breaking changes without plans for smooth evolution
- EDM4hep v1.0 planned to be finished soon
  - Backward compatible or transparent migration afterwards

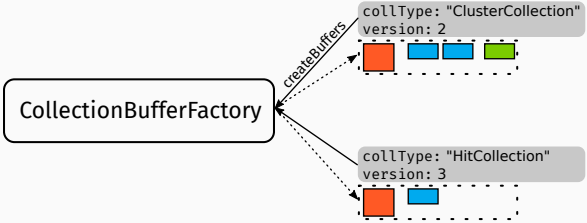


Supplementary Material

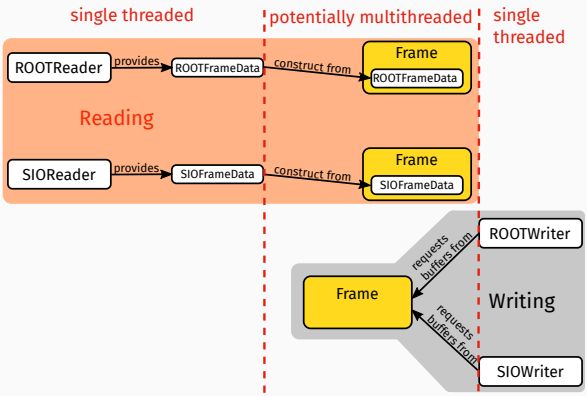
I/O low level basics

- I/O is based on collections
- `CollectionBuffer` holds all necessary data to (de)serialize a collection
  - Simple POD buffers (AoS)
  - I/O backend only needs to handle these
- `CollectionBufferFactory` creates empty buffers
  - (type, version) → `std::function`
  - Populated during datamodel library loading





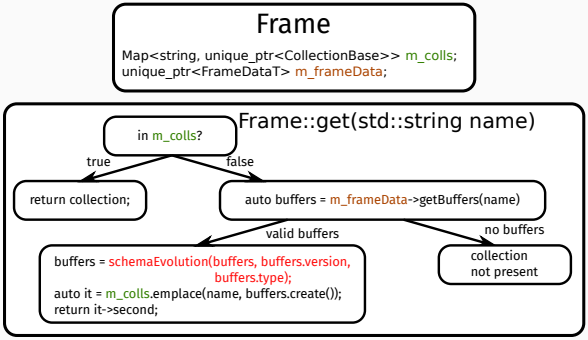
I/O on the Frame level



- Readers & Writers assumed to be single threaded
  - Low level building blocks
- Defer work as long as possible
  - Minimize time in Reader
- Frame can be constructed from "arbitrary" `FrameDataT`
  - Provides `CollectionBuffers`
  - Contain complete data for a Frame

Schema evolution - Technical details

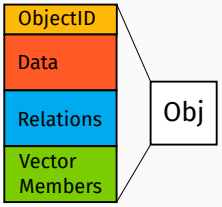
- Called as early as possible and as late as necessary
  - Earliest point where we have collection buffers from all backends is in Frame
- Schema evolution functions available from `SchemaEvolution` singleton
  - Populated during shared library loading
- Schema evolution can be a no-op



More recent transparent(-ish) changes

- Stable collection IDs
  - Initially: Insertion order into Frame
  - Now: Hash of collection name
  - 32 bits for transparent migration
- RNTuple based backend
- Storing datamodel definition in *metadata Frame*
  - Always possible to regenerate datamodel from datafile
  - Retrievable programmatically
  - Dumping via `podio-dump`
  - String literal embedded into binary

```
struct ObjectID {
  int index;
  uint32_t collectionID;
};
```




```
readelf -p .rodata libedm4hep.so | grep options
  [   368]  {"options": {<...>},
"schema_version": 1, "components": {<...>},
"datatypes": {<...>}}
```

podio - datamodel definition

```
components:
  edm4hep::Vector3f:
    Members: [float x, float y, float z]

datatypes:
  edm4hep::ReconstructedParticle:
    Description: "Reconstructed Particle"
    Author : "F.Gaede, DESY"
    Members:
      - edm4hep::Vector3f     momentum   // [GeV] particle momentum
      - std::array<float, 10> covMatrix  // energy-momentum covariance
    OneToOneRelations:
      - edm4hep::Vertex startVertex // start vertex associated to this particle
    OneToManyRelations:
      - edm4hep::Cluster clusters  // clusters that have been used for this particle
      - edm4hep::ReconstructedParticle particles // associated particles
    ExtraCode:
      declaration: "bool isCompund() const { return particles_size() > 0; }\n"

  edm4hep::ParticleID:
    VectorMembers:
      - float parameters // hypothesis params
```

*extracted from edm4hep.yaml

- Reusable components
- Fixed sized arrays as members
- *VectorMembers* for variable sized array members
- 1 – 1 and 1 – N relations
- Additional user-provided code

podio - features of generated code

```cpp
auto recos = ReconstructedParticleCollection();
// ... fill ...
for (auto reco : recos) {
  auto vtx = reco.getStartVertex();
  for (auto rp : reco.getParticles()) {
    auto mom = rp.getMomentum();
  }
}
```

← c++17 code with “value semantics”

↓ Python bindings via PyROOT

```python
recos = ReconstructedParticleCollection()
#... fill ...
for reco in recos:
  vtx = reco.getStartVertex()
  for rp in reco.getParticles():
    mom = rp.getMomentum()
```

```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()
```

← Using `RDataFrame` to read ROOT files (`uproot` also possible)

CMake interface for projects using podio

```cmake
find_package(PODIO)

# generate the c++ code from the yaml definition
PODIO_GENERATE_DATAMODEL(edm4hep edm4hep.yaml headers sources IO_BACKEND_HANDLERS "ROOT;SIO")
# compile the core data model shared library (no I/O)
PODIO_ADD_DATAMODEL_CORE_LIB(edm4hep "${headers}" "${sources}")
# generate and compile the ROOT I/O dictionary
PODIO_ADD_ROOT_IO_DICT(edm4hepDict edm4hep "${headers}" src/selection.xml)
# compile the SIOBlocks shared library for the SIO backend
PODIO_ADD_SIO_IO_BLOCKS(edm4hep "${headers}" "${sources}")

# Install the created targets
install(TARGETS edm4hep edm4hepDict edm4hepSioBlocks)
```

- Easy to use functions for integrating a podio generated EDM into a project
- Split into core EDM library and I/O handling for different backends
  - Pick what you need
  - I/O handling parts dynamically loaded by podio on startup