Status of EDM4hep

Towards a first stable release

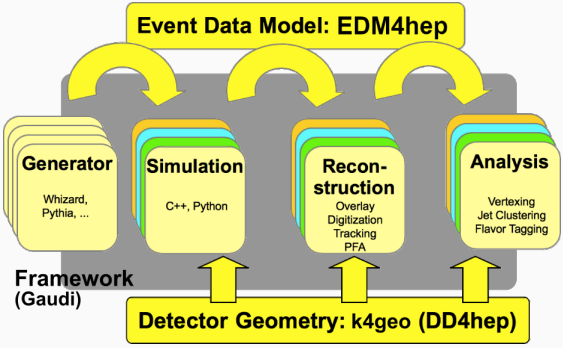
Thomas Madlener

ILD group meeting

May 07, 2024

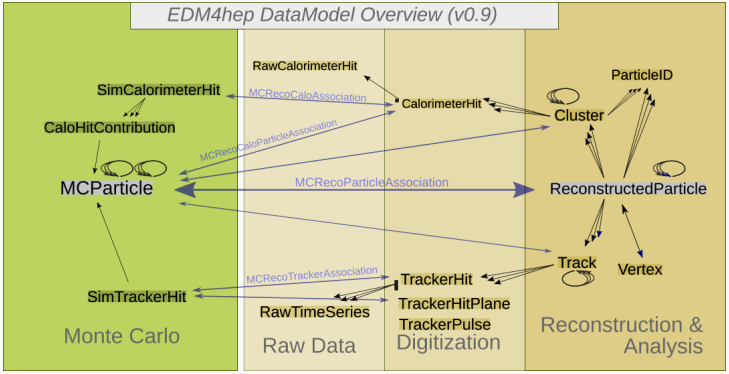
This project has received funding from the European Union's Horizon 2020 Research and Innovation programme under grant agreement No 101004761.

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

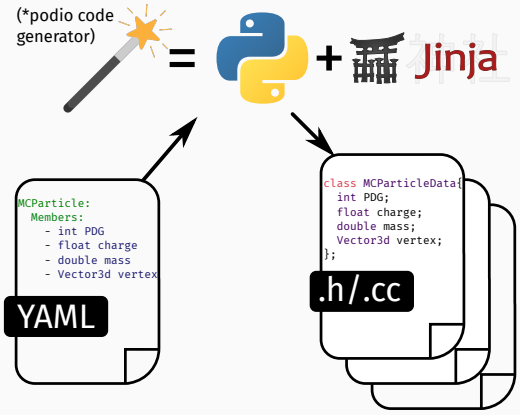
edm4hep.web.cern.ch

AIDASoft/podio

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

The podio EDM toolkit

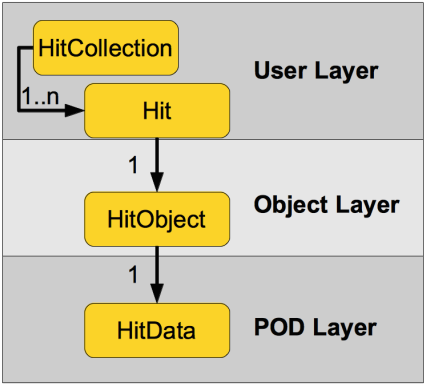
- Implementing a performant event data model (EDM) is non-trivial
- Use `podio` to generate code starting from a high level description
- Provide an easy to use interface to the users
- Main customers and feature drivers
  - key4hep/EDM4hep
  - eic/EDM4eic



AIDASoft/podio

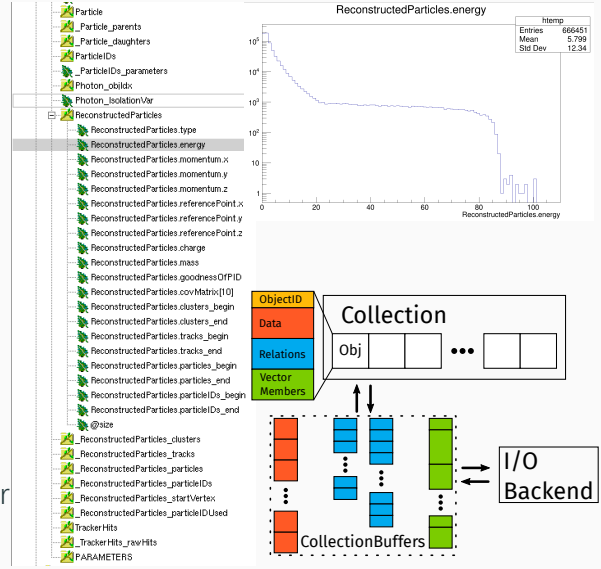
The three layers of podio

- 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**(!)
  - Can be used in `RDataFrame` (**FCCAnalyses**) or with `uproot`
  - Also with Julia
- Adding more I/O backends is possible
  - Alternative SIO backend exists
  - Working on `RDataSource` for better RDataFrame integration

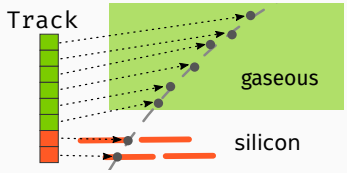


Schema evolution


```
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



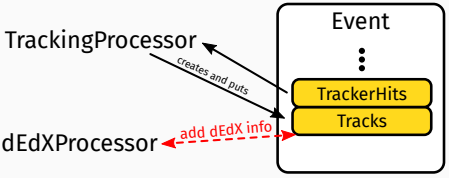

```yaml
interfaces:
  edm4hep::TrackerHit:
    Types: [edm4hep::TrackerHit3D, edm4hep::TrackerHitPlane]
    Members:
      - edm4hep::Vector3f position [mm] // hit position

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

Consistent mutability concept

- Some inconsistencies inherited from LCIO
  - Stricter multithreading concept in EDM4hep
- *Things can only be mutated during creation*
  - Trivial thread safety
  - Improved provenance
- Opportunity to “clean up” some things
- Some workflows cannot be directly ported

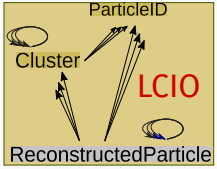


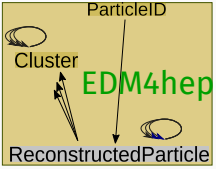

```
edm4hep::RecDqdx:
  Description: "dE/dx or dN/dx"
  Members:
    - edm4hep::Quantity dQdx  // value + error
  OneToOneRelations:
    - edm4hep::Track track    // computed from here
```

ParticleID handling EDM4hep vs LCIO

- Remove `ParticleID` relation from `Cluster`
  - Found no usage in ILD / CLIC reconstruction
- Make `ParticleID` have a one-to-one relation to `ReconstructedParticle`
  - Also remove `particleIDUsed`
- `ParticleID` has been (ab)used in LCIO as *transient parameter (values) store*
  - Will require change of pattern for EDM4hep
- Simple use cases become simpler with EDM4hep
- Tooling keeps the rest at the same level
- Some usability isses still to be addressed
  - keyhep/EDM4hep#298

key4hep/EDM4hep#268





ParticleID related utilities

- `edm4hep::utils::PIDHandler` similar to `UTIL::PIDHandler`
  - Get related `ParticleID`s from a `ReconstructedParticle`
  - Retrieve some PID metadata
  - Slightly more modern interface for EDM4hep
- Handling of necessary metadata very different
  - LCIO: *collection parameters* - tight coupling
  - EDM4hep: *file level metadata* - looser coupling
  - Gory details here and here
- See the documentation for more usage examples
- **Feedback very much appreciated!**

ParticleID handling comparison

Getting the dE/dx distance wrt an electron for all particles

```cpp
using namespace EVENT;
using namespace UTIL;

auto recos = event->getCollection("PandoraPFOs");
auto pidHandler = PIDHandler(recos);

const auto dEdxId = pidHandler.getAlgorithmID(_dEdxname);
const auto dEdx_e_Id = pidHandler.getAlgorithmID(dEdxId,
                                                 "e_dEdx_dist");

for (int i = 0; i < recos->getNumberOfElements(); ++i) {
  auto p = static_cast<ReconstructedParticle*>(
             recos->getElementAt(i));

  if (p->getCharge() == 0.0) {
    continue; // only charged particles have tracks
  }

  const auto& dEdxParams = pidHandler.getParticleID(p, dEdxId);
  const auto dEdx_e_dist = dEdxParams[dEdx_e_Id];

  // do something with the particle and the dEdx distance
}
```

LCIO

```cpp
using namespace edm4hep;
using namespace edm4hep::utils;

const auto dEdx = event.get<ParticleIDCollection>("dEdx");


const auto dEdxMeta = PIDHandler::getAlgoInfo(metadata, "dEdx");
const auto dEdx_e_Id = getParamIndex(dEdxMeta, "e_dEdx_dist");


for (const auto pid : dEdx) {

  const auto p = pid.getParticle();
  const auto dEdx_e_dist = dEdx.getParameters()[dEdx_e_Id];

  // do something with the particle and the dEdx distance
}
```

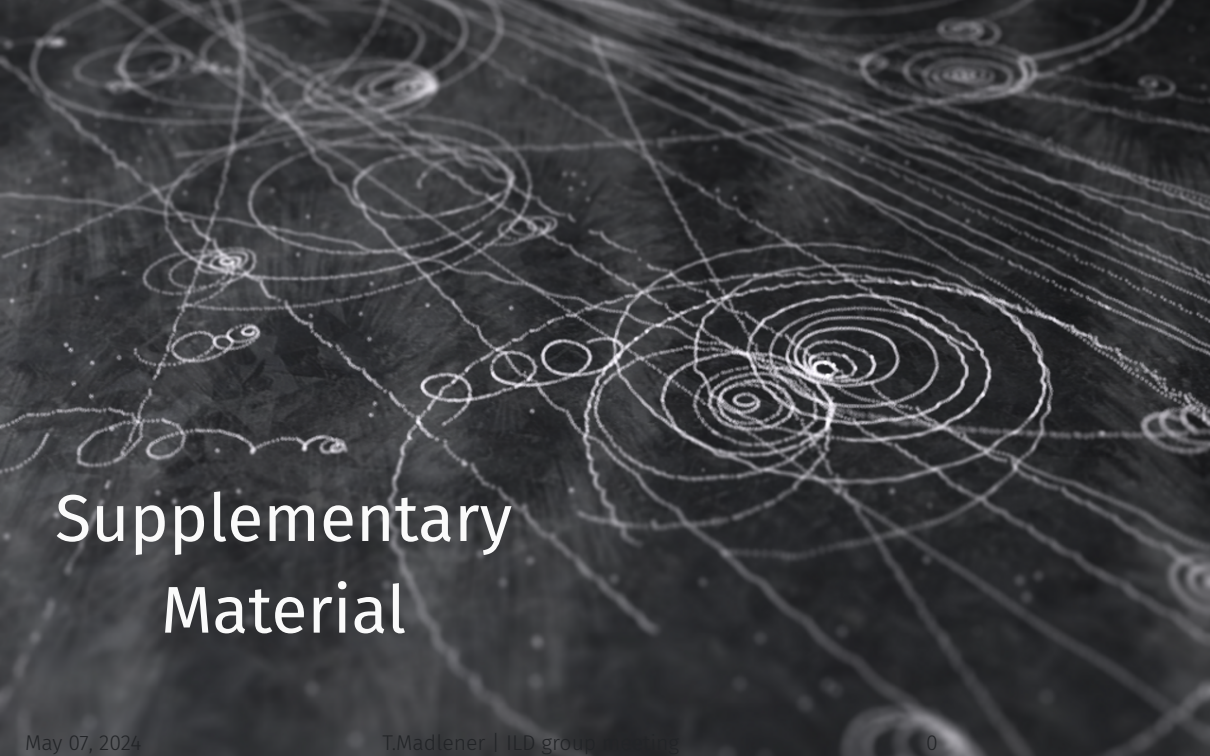
EDM4hep

Summary & Outlook

- EDM4hep is currently undergoing the final developments before v1.0
- podio as generating tool has accomodated the necessary feature requests
- Addressed some conceptual issues inherited from LCIO
  - `ParticleID`s & handling with largest differences
  - First version of corresponding utilities in place
- Transparent migrations and backwards compatibility after v1.0
- **Now is the best* time to test and request changes!**
  - Current version of EDM4hep available via the Key4hep nightlies
    `/cvmfs/sw-nightlies.hsf.org/key4hep/setup.sh`

---

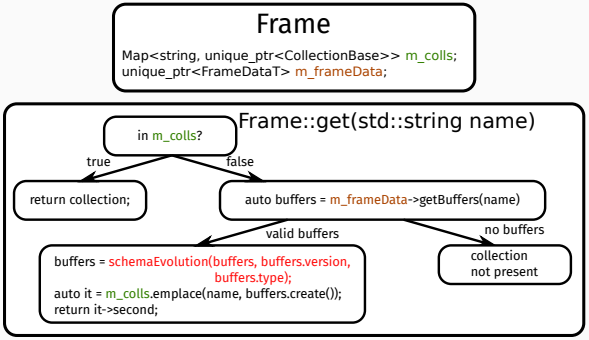
*last chance ;)



Supplementary Material

Schema evolution - Technical details

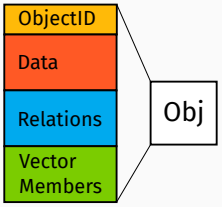
- Called as early as possible and as late as necessary
  - Earliest point where we have collection buffers from all backends is in Frame
- Schema evolution functions available from `SchemaEvolution` singleton
  - Populated during shared library loading
- Schema evolution can be a no-op



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

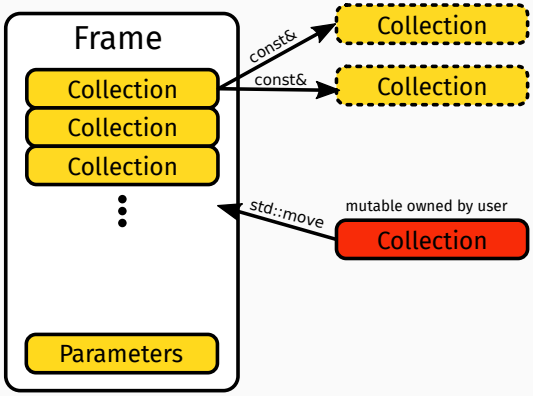
```
struct ObjectID {
  int index;
  uint32_t collectionID;
};
```




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

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);
```