Flexible and minimal-overhead Event Data Model for track reconstruction in ACTS

Paul Gessinger
CERN
2023-05-09 - CHEP 2023
Introduction
What is ACTS?

- Experiment-independent toolkit for track reconstruction applications
- Modern architecture and code, unit tested, continuous integration
- Minimal external dependencies
- Ready for multi-threading by design

Goals

- Provide established algorithms in a modern package
- Community platform for R&D across various experiment
- Provide testbed for R&D activities including new algorithms, machine learning, heterogeneous computing
Evaluation and/or deployment by multiple experiments

Performance with ideal geometry

- Pions embedded in
  - AuAu with 50 kHz pileup
  - pp with 3 MHz pileup

- All embedded tracks:
  - > 2 MVTX hits
  - > 20 TPC hits

Lower right:
Upsilon(1S) mass resolution versus centrality

- AuAu with 50 kHz pileup

$T_p$ vs. Eff.

Impact parameter [fm]

Mass resolution [MeV]

Simulation Internal sPHENIX (1S)ϒ

& FASER-2

ePIC (EIC) (presentation + keynote yesterday)

STCF (presentation just now)
Event Data Model
## ACTS Event Data Model

- **Event Data Model (EDM) is critical piece of reconstruction software**

### Internal EDM
- Data objects to pass around between different parts of the library
- Library specific, tightly coupled to the algorithm

### Public EDM
- Data objects clients directly interact with
- Should be experiment agnostic
- Extensible by experiment, easy integration
ACTS Event Data Model

- Event Data Model (EDM) is critical piece of reconstruction software

**Internal EDM**

- Data objects to pass around between different parts of the library
- Library specific, tightly coupled to the algorithm

**Focus so far!**
**ACTS Event Data Model**

- Event Data Model (EDM) is critical piece of reconstruction software

**Focus now!**

---

**Public EDM**

- Data objects clients directly interact with
- Should be experiment agnostic
- Extensible by experiment, easy integration

---
Client-focused primary output of tracking, used by track finding and fitting

Track-level quantities + sequence of track states with intermediate information

Tracks:
- Defining parameters at perigee
- Global track quantities ($\chi^2$, num. hits, holes, outliers, etc)

Track states:
- Local parameters + cov. at geometric object (e.g. sensor)
- Calibrated measurement, dimension, covariance
- Auxiliary information like jacobian, type flags, etc
Architecture
**Architecture**

- **Container** is the primary data object
- Elements in containers are thin views (proxies) into them
- Container and proxy provide user-facing API
- Want this to be fully integratable into experiment IO infrastructure!

```cpp
for (auto track : tracks) {
    track.parameters() = Acts::BoundVector::Zeros();
    track.nMeasurements() = 42;
    for (auto trackState: track.trackStates()) {
        trackState.referenceSurface();
    }
}
```

![Diagram](Image)
Architecture

- Track finding
- Track fitters
- TrackProxy
- TrackContainer
- TrackState backend
- Track backend

Performance monitoring

Downstream reconstruction

backend interface / contract

EDM4hep input

EDM4hep output

Experiment agnostic: decouple interface from storage implementation

Conversion (lossy)

Transparent, direct backend
Architecture

- Track finding
- Track fitters
- Performance monitoring
- Downstream reconstruction
- TrackProxy
  - TrackContainer
  - TrackState backend
  - Track backend

Experiment agnostic: decouple interface from storage implementation
Conversion (lossy)
Transparent, direct backend
Architecture

- Track finding
- Track fitters
- Performance monitoring
- Downstream reconstruction

TrackProxy
- TrackContainer

TrackState backend
- Track backend

backend interface / contract

Experiment agnostic: decouple interface from storage implementation
Architecture

- Track finding
  - TrackProxy
  - TrackContainer

- Track fitters

- Performance monitoring

- Downstream reconstruction

- TrackState backend
  - PODIO
  - ATLAS xAOD

- Track backend
  - std::vector
  - PODIO
  - ATLAS xAOD

Backend interface / contract

Experiment agnostic: decouple interface from storage implementation
Implementation
**Fully decoupled interface** seen by ACTS and client consumers from the backend implementation: **Backend can be fully experiment-specific**

- **First attempt**: inheritance and virtual function calls: resulted in **undesirable overhead**
- **Second attempt**: template based extension, **negative overhead** (likely due to better optimization)

- Supports dynamically added columns (if the backend supports it)

![Graph showing performance comparison between main, virtual interface, and template interface for CKF tracks and Truth tracks.](image-url)
Backend interface / contract

- Interface-layer expects backend to implement set of methods
- Component access largely via single function and compile-time hashes of component names
- Dedicated methods where backends needs flexibility for implementation
- Design goal: allow ACTS components to directly manipulate the backend storage

```cpp
track.parameters() = Acts::BoundVector::Zeros();
```

Core requirements

- Backend can return (non-dangling) references to memory representation
- Tracks and track states can be fully identified by an index
- Track states parameters, covariances + jacobians are in an indexed container somewhere
Common EDM package for the key4hep software stack

Built using PODIO framework: common definition of various data types, relationships

Contain `edm4hep::Track` & `edm4hep::TrackState`

- Uses the LCIO parametrization \( d_0, z_0, \phi, \tan \lambda, \Omega \) (ACTS uses \( l_0, l_1, \phi, \theta, q/p, t \))
- Track states are described using perigee parameters only (ACTS uses varying local parametrization + link to geometry object)

Direct & transparent backend in EDM4hep not feasible

- Required contract cannot be fulfilled:
  - No stable references to native parametrization
  - Loss of on-surface hit position

- Instead: **Full (lossy) conversion** to and from EDM4hep tracks implemented (and in turn is backend agnostic)
Architecture

- **EDM4hep input**
  - Track finding
  - TrackProxy
  - TrackContainer
  - Track fitters

- **Backend interface / contract**
  - TrackState backend
    - std::vector
    - PODIO
    - ATLAS xAOD
  - Track backend
    - std::vector
    - PODIO
    - ATLAS xAOD

- **Performance monitoring**
- **Downstream reconstruction**

**Conversion (lossy)**

Paul Gessinger
2023-05-09 - CHEP 2023
Architecture

**Track finding**

**Track fitters**

**Performance monitoring**

**Downstream reconstruction**

**EDM4hep input**

**TrackProxy**

**TrackContainer**

**TrackState backend**

**Track backend**

**PODIO**

**ATLAS xAOD**

**Conversion (lossy)**

**Transparent, direct backend**

**Backend interface / contract**
Goal

- Demonstrate **ability to integrate** with an external IO solution like PODIO
- This is not an alternative to EDM4hep, but help us understand requirements

- Specify ACTS EDM in **PODIO**-yaml \(^1\) in *plugin*
- Implemented **ActsPodioEdm::Track** + **ActsPodioEdm::TrackStates**
  - Use *components* to produce stable references to fulfill backend contract
  - Auxiliary data types for dense columns overallocated storage for measurements
  - Experiment-aware translation helper for surfaces and uncalibrated measurements

- **Full IO roundtrip implemented and tested, Kalman Filter can run on this without modifications**

\(^1\)see also talk on PODIO on Thursday
Summary & conclusion

- ACTS has gained client-facing *high-level* Track Event Data model!
- Track finding + fitting already produce this data type (generic refitting pending)
- Interface layer is **fully separated** from backend implementation
- Backend allows **direct integration with experiment** IO framework
- Support conversion to and from EDM4hep for Tracks
- Implemented custom PODIO-based EDM demonstrator
  - Transparent backend with PODIO feasible

Further work

- Migrate all downstream tools to work on Track EDM
- Characterize PODIO backend performance
Backup
Experiment interface

- **PODIO** backend still supposed to be experiment agnostic
- Experiment-knowledge needed to persist otherwise transient information

### Surfaces

- Two types: part of detector geometry, *ad-hoc* surfaces
- Encode known surfaces as identifiers, serialize *ad-hoc* surfaces
- Make no assumptions on identification model

### Measurements

- ACTS uses strong type-erasure for experiment-specific input measurements
- Cannot serialize type-erased measurements automatically

- Factorized to experiment-specific helper class to implement these conversions