Event Data Model: Upcoming developments

Paul Gessinger
CERN
2022-09-27 - ACTS Workshop

(Brief) EDM overview

- Spacepoints: rely on experiment specific SPs + adapter functions
- Seeds: `template <typename SpacePoint> class Seed;`
- Measurements
  - **Uncalibrated**: experiment/subdetector specific, not actually exposed to ACTS
    - *SourceLink*: proxy for the uncalibrated measurement
  - **Calibrated**[1]: measurement position + covariance + link to sensitive element(up to 6D)
  - Fully experiment specific **calibrator** turns SourceLinks (i.e. uncalibrated meas.) into latter exploiting any info needed during track finding / fitting
- Track state container (called *MultiTrajectory* in ACTS)
- Track: currently only ad-hoc implementation for standalone performance calculation

[1]Technically: one single representation, one representation on track state

Uncalibrated Measurement EDM

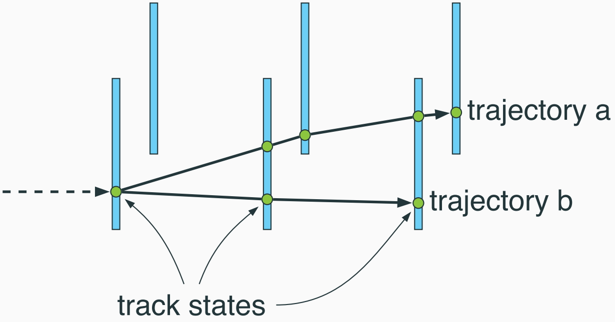
- This is not actually ACTS specific
- Modelled closely after experiment readout structure
- ACTS only operates on these via experiment-specific *calibrator* object
  - Turns experiment-specific uncalibrated measurement into ACTS-readable calibrated measurement

Track state EDM + Track EDM

- ACTS so far focused on **internal** EDM for interchange between ACTS components
  - Therefore: no `Acts::Track` so far!
- Goal: develop ACTS based replacement for `Trk::Track`
  - Provide necessary information, access to track states, etc.
  - Allow for refit with as little overhead as possible
  - Support different fitters, clients
  - Good integration with ATLAS EDM infrastructure

Track State Container: `Acts::MultiTrajectory`

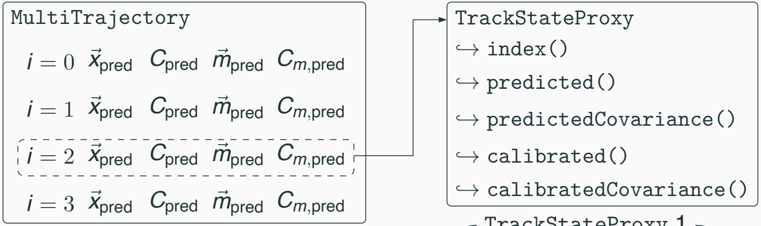
- MultiTrajectory is the track state container that the fitters use
- Want to build `Acts::Track` on top of it
- CKF uses a *buffer* internally to collect TS candidates before selection



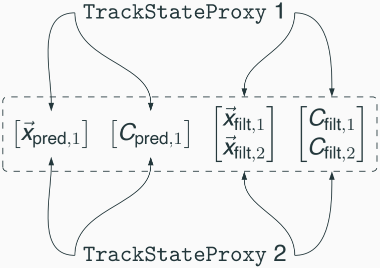
- CKF branches:
  - TrackStates are connected by a *previous* index
  - MultiTrajectory was designed to reduce memory usage

MultiTrajectory

- MultiTrajectory: column based track state container
- TrackStateProxy: object oriented view on top of an index (much like AuxElement)



- MultiTrajectory uses **dense storage** for covariances, parameter vectors
- Track states are like a *sparse* collection on top of these
  - Want ability to **share** components between track states.



Backend abstraction

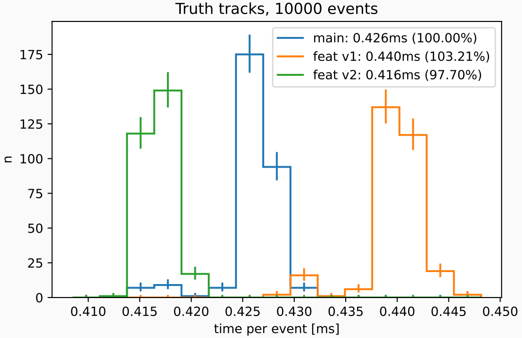
- Storage backend fully abstracted from container API
- Backend needs to support all operations used by the fitters (but implementation detail how)
- Component access implemented via access methods, compile-time hashed string and switch/case
  - Limits API surface required from backends
  - Method to check if a component is set:

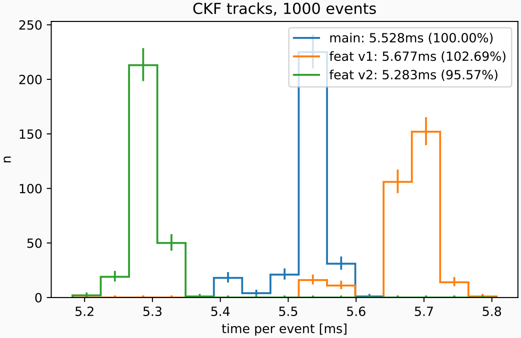
    ```
    template <HashedString key> constexpr bool has() const;
    ```
  - Method to access component generically:

    ```
    template <typename T, HashedString key> constexpr T& component();
    ```
  - + a few more
- CKF, KF and GSF now reuse single MultiTrajectory

Runtime performance

- Benchmarked using ACTS truth tracking and track finding examples, single thread, 300 runs reach
- Initial implementation with virtual functions slower, updated implementation CRTP actually faster than before
- All give bit identical output





Memory footprint

- High activity events: CKF produces **a lot** of track states
  - Pythia8 ttbar $\mu = 200$ event CKF: 2GB non-measurement states + 1GB measurement states
  - $\rightarrow$ truth tracking (best case): 40MB non-measurement states + 60MB measurement states
- Partly because of algorithm details (holes + material states as track states), partly because of design
- Work ongoing to optimize without algorithm changes
  - Switched to jagged structure for measurement (varying dimension)
  - Evaluate / switch storing coveriances as triangles
  - If need be: switch to `float` for covariance storage
- Algorithm changes: e.g. use buffer MultiTrajectory to filter earlier

Extra columns I

- Hashed key based component access nicely supports *dynamic extra column*
- Ideally: map 1:1 to xAOD decorations
- Allows adding fitter specific extra data to track states (like GSF number of components)
- Would enable a `Acts::Track` to be very flexible:
  - Minimal set of generic properties in ACTS standalone
  - Additional fitter- or experiment-specific information can be attached and preserved
- Have minimal standalone prototype + proof-of-concept for xAOD decorations: no roadblocks discovered

Considerations on `Acts::Track` + TrackContainer

- Intend to model this largely after the ATLAS track class
- API surface is *reasonably small*, stored data $\sim$ maps to ACTS data
  - Perigee
  - Track states (on surface)
  - Outliers ($\rightarrow$ track states in ACTS)
  - Track summary (potentially special muon track summary) $\rightarrow$ want to replace with dynamic columns if possible
  - Fit quality
  - Track info $\rightarrow$ want to replace with dynamic columns if possible
- Want to avoid conversion if at all possible
- Hope that dynamic columns can help us make this flexible

Development plan

- Minimal Track prototype exists, but is not ready for prime time
- Build out minimal set of information to replace the `Trajectory` type in the examples
  - Begin with building Track instead of Trajectory outside of fitters (temporarily)
- Update all clients of `Trajectory` to use the Track class
  - Tests a minimal but reasonably comprehensive workflow
- Deploy minimal version, test in ATLAS (has specific requirements)
- Update / extend fitter interfaces to return Track and allow Track refitting

Backup

Prototype for backend abstraction I

- Presented last week in ACTS-ITk meeting
  - Updated using CRTP now
- Step 1: make MultiTrajectory (pseudo-)abstract
  - Adjust fitters to go through a pointer to it
  - Right now: default implementation which works $\sim$ as before:

```
class VectorMultiTrajectory final
  : public MultiTrajectory<VectorMultiTrajectory>;
```

  - Change Propagator to allow passing in a result: needed to allow clients to construct the MultiTrajectory beforehand

```
template <...> Result<...> propagate(
  const parameters_t& start,
  const propagator_options_t& options,
  result_t inputResult) const; // <- contains MultiTrajectory instance
```

- Also want to allow e.g. CKF to reuse *one* MultiTrajectory for all seeds (not done yet)

Prototype for backend abstraction II

- Step 2: Prevent external access to index data structure, add interfaces to replace
  - Internal component access via CRTP interface to backend using strings hashed at **compile-time**: `using HashedString = std::uint32_t;`
  - Method for unsetting components:
    ```
    void unset(TrackStatePropMask target)
    ```
  - Method to check if a component is set:
    ```
    template <HashedString key>
    constexpr bool has() const;
    ```
  - Method to access component generically:
    ```
    template <typename T, HashedString key>
    constexpr T& component();
    ```
  - Method for component sharing:
    ```
    ts.shareFrom(first, PM::Predicted);
    ```
    instead of
    ```
    ts.data().ipredicted = first.data().ipredicted;
    ```

Prototype for backend abstraction III

- Step 3: Refactor internal index access to support type-erasure
  - Two kinds of components: with (optional: predicted, filtered, smoothed, calibrated, jacobian) and without indirection (all track states have one)
  - Indirect components can be stored in an extra container to save memory (want to use this in xAOD)

```
auto ts = traj.addTrackState();
ts.predicted(); // is equivalent to
ts.component<Parameters, "predicted"_hash>(); // can also be supplied as argument
ts.predicted() = makePredicted();
ts.component<Parameters, "predicted"_hash>() = makePredicted();
ts.hasPredicted();
ts.has<"predicted"_hash>();
```

  - *Public* interface of MultiTrajectory / TrackStateProxy stays the same!
  - *Naturally* supports dynamic additional columns

Prototype for backend abstraction IV

```
std::any component_impl(HashedString key, IndexType istate) {
  using namespace Acts::HashedStringLiteral;
  switch (key) {
    case "previous"_hash: return &m_index[istate].iprevious;
    case "predicted"_hash: return &m_index[istate].ipredicted;
    //   ^- + extra lookup methods for values + cov
    // ...
    case "chi2"_hash: return &m_index[istate].chi2;
    default:
      auto it = m_dynamic.find(key);
      if (it == m_dynamic.end()) { throw std::runtime_error("x");}
      auto& col = it->second;
      assert(col && "Dynamic column is null");
      return col->get(istate);
  }
}
```

Prototype for backend abstraction V

- Step 4: Remove internal index access
  - Everything goes through the interface
  - MultiTrajectory base class and clients know nothing about storage implementation

Backend interface

```
TrackStateProxy::Parameters parameters_impl(IndexType); // + const
TrackStateProxy::Covariance covariance_impl(IndexType); // + const
TrackStateProxy::Covariance jacobian_impl(IndexType); // + const
TrackStateProxy::Measurement measurement_impl(IndexType); // + const
TrackStateProxy::MeasurementCovariance measurementCovariance_impl(IndexType); // + const
std::size_t addTrackState_impl(TrackStatePropMask mask,size_t iprevious);
void shareFrom_impl(IndexType iself, IndexType iother,
                    TrackStatePropMask shareSource,
                    TrackStatePropMask shareTarget);
void unset_impl(TrackStatePropMask target, IndexType istate);
constexpr bool has_impl(HashedString key, IndexType istate) const;
std::size_t size_impl() const;
void clear_impl();
std::any component_impl(HashedString key, IndexType istate);
std::any component_impl(HashedString key, IndexType istate) const;
template <typename T>
constexpr void addColumn_impl(HashedString key);
constexpr bool hasColumn_impl(HashedString key) const;
```

Build performance

- Had to untemplate components using MultiTrajectory like `GainMatrixUpdater`, `GainMatrixSmoother`, `MeasurementSelector`
- Strongly increases memory consumption in the build
- **Mitigated** by refactoring and moving Eigen instantiations into seperate CUs
- E.g. the following section takes ∼**600MB** to compile

```
const auto H = projector.template topLeftCorner<kMeasurementSize, eBoundSize>().eval();
ParametersVector res = calibrated - H * predicted;
return (res.transpose() * ((calibratedCovariance +
  H * predictedCovariance * H.transpose())) .inverse() * res).eval()(0, 0);
```

- Needs to be done carefully to preserve performance

Build performance

file	max_rss_main	max_rss_feat_v2	relative
CombinatorialKalmanFilterError.cpp	1.28614	1.24109	96.4968
MeasurementSelector.cpp	864.469	879.059	101.688
GainMatrixUpdater.cpp	1280.87	1300.28	101.515
GainMatrixSmoother.cpp	513.376	487.125	94.8865
GsfUtils.cpp	628.855	609.251	96.8827
TrackFindingAlgorithmFunction.cpp	794.558	765.542	96.3482
TrackFittingAlgorithmFunction.cpp	818.065	798.306	97.5847
AlignmentAlgorithmFunction.cpp	1369.24	1329.99	97.133
CombinatorialKalmanFilterTests.cpp	1087.35	1230.05	113.123
GsfTests.cpp	1642.13	1686.95	102.729
KalmanFitterTests.cpp	1232.11	1282.43	104.085
GainMatrixUpdaterTests.cpp	383.144	437.797	114.264
GainMatrixSmootherTests.cpp	329.216	413.655	125.649
GsfComponentMergingTests.cpp	455.721	440.185	96.5909