IRT Integration status

C. Chatterjee

and C. Dilks

Dependence of the IRT algorithm to eicd:

The IRT algorithm depends on the eicd (eic data model).
The IRT has two primary modules.

a) Reconstruction (Single Cherenkov Photon angle)

b) Evaluation (Evaluation of hypothesis of a particle (MC or Reco))

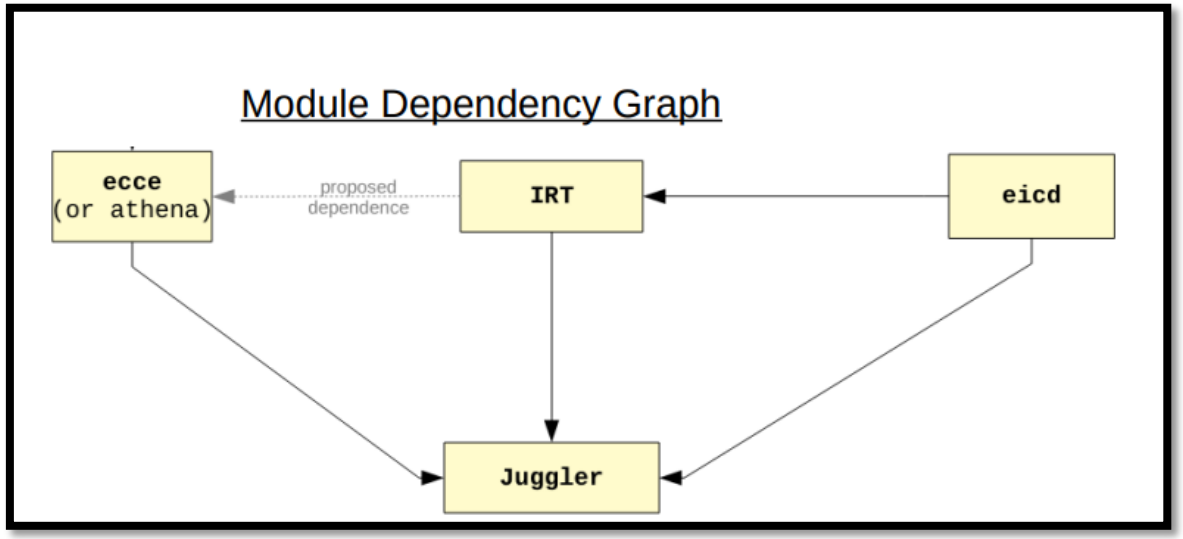
Both steps depend on the EIC data Structure.

Dependence of the Juggler algorithm to eicd:

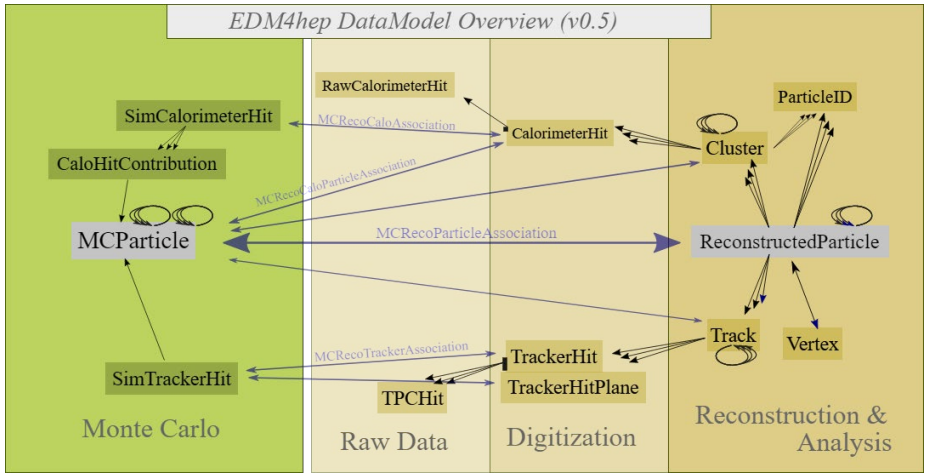
The Juggler depends in many foldes:

a) The IRT part within juggler (eicd dependence)

b) Other associations (JugTrack,JugReco ..) in juggler to IRT (internal eicd dependence).



There are some significant changes compared to winter 2021 in the data framework. The most relevant one is related to the transition towards OneToOneRelation/OneToManyRelations instead of eic::Index → edm4hep



Change in [eic_data.yaml]

```
eic::CherenkovParticleID:
  Description: "Cherenkov detector PID"
  Author: "A. Kiselev, C. Dilks"
  Members:
    - eic::Index        ID                    // Unique entry ID
    - eic::Index        recID                 // Index of the associated ReconstructedParticle particle, if any
  VectorMembers:
    - eic::CherenkovPdgHypothesis options        // Evaluated PDG hypotheses, typically (e/pi/K/p)
    - eic::CherenkovThetaAngleMeasurement angles // Evaluated Cherenkov angles for different radiators
```

Data model IRT can work with

```
eicd::CherenkovParticleID:
  Description: "Cherenkov detector PID"
  Author: "A. Kiselev, C. Dilks"
  VectorMembers:
    - eicd::CherenkovPdgHypothesis options        // Evaluated PDG hypotheses, typically (e/pi/K/p)
    - eicd::CherenkovThetaAngleMeasurement angles // Evaluated Cherenkov angles for different radiators
  OneToOneRelations:
    ## @TODO: should it be one-to-one?
    - eicd::ReconstructedParticle associatedParticle // associated reconstructed particle
```

Data model Now

```
62
63      // Then the Cherenkov-to-reconstructed mapping; FIXME: may want to use Cherenkov-to-simulated
64      // mapping to start with, for the debugging purposes;
65      std::map<eic::Index, const eic::CherenkovParticleIDData*> rc2cherenkov;
66      for(const auto &pid: *cherenkov)
67        rc2cherenkov[pid.recID] = &pid;
68
69      // Loop through all MC tracks;
70      for(auto mctrack: *mctracks) {
71        // FIXME: consider only primaries for now?;
72        if (mctrack.g4Parent) continue;
73
74  #ifdef _USE_RECONSTRUCTED_TRACKS_
75        // Find a matching reconstructed track;
76        auto rctrack = mc2rc.find(mctrack.ID) == mc2rc.end() ? 0 : mc2rc[mctrack.ID];
77        if (!rctrack) continue;
78
79        // Find a matching Cherenkov PID record;
80        auto cherenkov = rc2cherenkov.find(rctrack.ID) == rc2cherenkov.end() ? 0 : rc2cherenkov[rctrack.ID];
81  #else
82        auto cherenkov = rc2cherenkov.find(mctrack.ID) == rc2cherenkov.end() ? 0 : rc2cherenkov[mctrack.ID];
83  #endif
84        if (!cherenkov) continue;
85
86        // Loop through all of the mass hypotheses available for this reconstructed track;
87        {
88          const eic::CherenkovPdgHypothesis *best = 0;
89
90          //printf("%d %d\n", cherenkov->options_begin, cherenkov->options_end);
91          for(unsigned iq=cherenkov->options_begin; iq<cherenkov->options_end; iq++) {
92            const auto &option = (*options)[iq];
93
94            // Skip electron hypothesis; of no interest here;
95            if (abs(option.pdg) == 11) continue;
96
97            if (!best || option.weight > best->weight) best = &option;
98            printf("radiator %3d (pdg %5d): weight %7.2f, npe %7.2f\n",
99                   option.radiator, option.pdg, option.weight, option.npe);
100         } //for
101         printf("\n");
102
```

IRT (code example)

We need to refactor the IRT code and all possible dependencies downstream to be compatible with edm4hep adaptations.

→ Several to-and-fro changes to stay up-to-date with upstream changes are foreseen.

ANY EXAMPLES KNOWN WHERE CHANGES ARE APPLIED IN THIS DIRECTION?

Change in [eic_data.yaml] (left: now; right: what IRT/(our juggler) is assuming). For reconstructed particle as an example

```
## ============================================================================
## Particle info
## ============================================================================

eicd::BasicParticle:
  #DEPRECATED
  Description: "Basic particle used internally to communicate basic particle properties."
  Author: "W. Armstrong, S. Joosten"
  Members:
    - eicd::Vector3f    p               // Momentum [GeV]
    - eicd::Vector3d    v               // Vertex [mm]
    - float             time            // Time in [ns]
    - int32_t           pid             // Particle PDG code
    - int16_t           status          // Status code
    - int16_t           charge          // Particle charge (or sign)
    - float             weight          // Particle weight, e.g. from PID algorithm [0-1]

eicd::ReconstructedParticle:
  Description: "EIC Reconstructed Particle"
  Author: "W. Armstrong, S. Joosten, F. Gaede"
  Members:
    - int32_t           type            // type of reconstructed particle. Check/set collection parameters ReconstructedParticleTypeNames and ReconstructedPart
    - float             energy          // [GeV] energy of the reconstructed particle. Four momentum state is not kept consistent internally.
    - eicd::Vector3f    momentum        // [GeV] particle momentum. Four momentum state is not kept consistent internally.
    - eicd::Vector3f    referencePoint  // [mm] reference, i.e. where the particle has been measured
    - float             charge          // charge of the reconstructed particle.
    - float             mass            // [GeV] mass of the reconstructed particle, set independently from four vector. Four momentum state is not kept consis
    - float             goodnessOfPID   // overall goodness of the PID on a scale of [0;1]
    - eicd::Cov4f       covMatrix       // cvariance matrix of the reconstructed particle 4vector (10 parameters).
    ##@TODO: deviation from EDM4hep: store explicit PDG ID here. Needs to be discussed how we
    ##       move forward as this could easilly become unwieldy without this information here.
    ##       The only acceptable alternative would be to store reconstructed identified
    ##       particles in separate collections for the different particle types (which would
    ##       require some algorithmic changes but might work. Doing both might even make
    ##       sense. Needs some discussion, note that PID is more emphasized in NP than
    ##       HEP).
    - int32_t           PDG             // PDG code for this particle
    ## @TODO: Do we need timing info? Or do we rely on the start vertex time?
  OneToOneRelations:
    - eicd::Vertex      startVertex     // Start vertex associated to this particle
    - eicd::ParticleID  particleIDUsed  // particle ID used for the kinematics of this particle
  OneToManyRelations:
    - eicd::Cluster     clusters        // Clusters used for this particle
    - eicd::Track       tracks          // Tracks used for this particle
    - eicd::ReconstructedParticle particles // Reconstructed particles that have been combined to this particle
    - eicd::ParticleID  particleIDs     // All associated particle IDs for this particle (not sorted by likelihood)
  ExtraCode:
    declaration: "
      bool isCompound() const {return particles_size() > 0;}\n
```

```
## ============================================================================
## Particle info
## ============================================================================

eic::BasicParticle:
  Description: "Basic particle used internally to communicate basic particle properties."
  Author: "W. Armstrong, S. Joosten"
  Members:
    - eic::Index        ID              // Unique particle index
    - eic::VectorXYZ    p               // Momentum [GeV]
    - eic::VectorXYZ    v               // Vertex [mm]
    - float             time            // Time in [ns]
    - int32_t           pid             // Particle PDG code
    - int16_t           status          // Status code
    - int16_t           charge          // Particle charge (or sign)
    - eic::Weight       weight          // Particle weight, e.g. from PID algorithm [0-1]

eic::ReconstructedParticle:
  Description: "EIC Reconstructed Particle"
  Author: "W. Armstrong, S. Joosten"
  Members:
    - eic::Index        ID              // Unique particle index
    - eic::VectorXYZ    p               // Momentum vector [GeV]
    - eic::VectorXYZ    v               // Vertex [mm]
    - float             time            // Time in [ns]
    - int32_t           pid             // PID of reconstructed particle.
    - int16_t           status          // Status code
    - int16_t           charge          // Particle charge (or sign)
    - eic::Weight       weight          // Particle weight, e.g. from PID algorithm [0-1]
    - eic::Direction    direction       // Direction (theta/phi of this particle [mrad])
    - float             momentum        // particle 3-momentum magnitude [GeV]
    - float             energy          // Particle energy, consistent with PID assigment [GeV]
    - float             mass            // The mass of the particle in [GeV]
    - eic::Index        vertexID        // Start vertex for this particle
    - eic::Index        trackID         // Index of the associated track, if any
    - eic::Index        ecalID          // Index of associated pos/barrel/neg ECAL cluster, if any
    - eic::Index        hcalID          // Index of associated pos/barrel/neg HCAL cluster, if any
    - eic::Index        cherID          // Index of associated pos/barrel/neg Cherenkov info, if any
    - eic::Index        tofID           // Index of the associated TOF info, if any
    - eic::Index        mcID            // Index of the associated MC particle, if any
```

```
37      -   bool GetCrossing(const ParametricSurface *surface, const eicd::TrajectoryPoint *point,
        36  +   double GetDistance(const ParametricSurface *surface, const eicd::TrackPoint *point) const;
        37  +   bool GetCrossing(const ParametricSurface *surface, const eicd::TrackPoint *point,
38      38                       TVector3 *crs) const;
39      -   TVector3 GetLocation(const eicd::TrajectoryPoint *point) const;
40      -   TVector3 GetMomentum(const eicd::TrajectoryPoint *point) const;
        39  +   TVector3 GetLocation(const eicd::TrackPoint *point) const;
        40  +   TVector3 GetMomentum(const eicd::TrackPoint *point) const;
```

Some fixes are made in the juggler (dRICH) is fixed on the JugTrack dependencies. We will test them later.

Other ongoing activities:

a. geometry integration (C. Dilks)

b. More realistic SiPM description (Ajit)