FD-CAFMaker update

Pierre Granger

Some context

Overhaul of the CAF format by Jeremy Wolcott

- Richer true interaction informations
- Hierarchical structure with common fields among all detectors and specific branches for specific detectors
- MCTruth particles are saved
- Reco particles/tracks/showers are saved

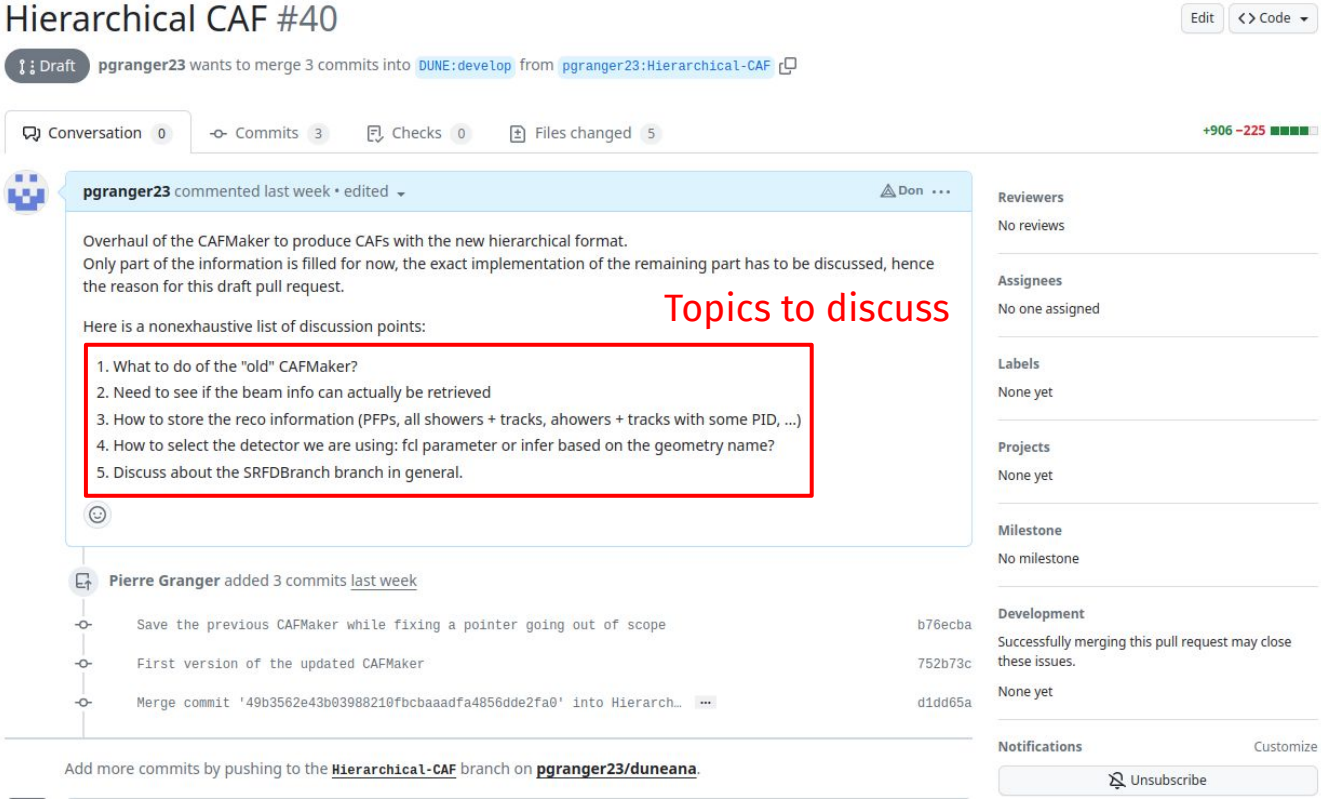
```
class StandardRecord
{
    public:
    /// Metadata about the detectors
    SRDetectorMetaBranch meta;
    /// Information about the beam configuration and beam pulse for this event
    SRBeamBranch beam;
    /// Truth information
    SRTruthBranch mc; ← Need to add info for atmo HERE
    /// Reconstructed info expected to be common to all (?) detectors
    SRCommonRecoBranch common;
    /// Reconstructed info unique to the FDs
    SRFDBranch fd; ← Need to add info for atmo HERE
    /// Reconstructed info unique to the ND complex
    SRNDBranch nd;
};
```

- In use for ND studies. Wanna follow to use common format.
- Also provides some fields required for the atmospherics that were not available before.

Draft pull request in duneana

First initial implementation of the new CAFMaker

https://github.com/DUNE/duneana/pull/40



CAFMaker versioning

Main options:

1. Getting rid of the previous CAFMaker version
2. Keep supporting the previous CAF format in the new CAFMaker
3. Keeping the previous CAFMaker as a CAFMaker_legacy module

-> Might be tricky to keep both as they require different duneanaobj versions!

Fixed a bug in the CAFMaker that (luckily) did not seem to have an impact before (lead to segfaults with the newer CAFMaker).

SetBranchAddress to pointer going out of scope

```
caf::StandardRecord sr;

if(fTree){
  caf::StandardRecord* psr = &sr;
  fTree->SetBranchAddress("rec", &psr);
}
```

Some fields are not filled

Beam info

```
//TODO: Need to see if these info can be retrieved
// inter.baseline ///< Distance from decay to interaction [m]
// inter.prod_vtx ///< Neutrino production vertex [cm; beam coordinates]
// inter.parent_dcy_mom ///< Neutrino parent momentum at decay [GeV; beam coordinates]
// inter.parent_dcy_mode ///< Parent hadron/muon decay mode
// inter.parent_pdg ///< PDG Code of parent particle ID
// inter.parent_dcy_E ///< Neutrino parent energy at decay [GeV]
// inter.imp_weight ///< Importance weight from flux file
```

Not sure any of this can actually be retrieved. Does someone know?
I imagine this could be used for some uncertainty treatment

Matching right detector

```
// full FDs (Phase II modules TBD...)
SRDetectorMeta fd_hd;    /// Horizontal drift (a.k.a. module 1)
SRDetectorMeta fd_vd;    /// Vertical drift (a.k.a. module 2)

// FD prototypes  (add VD ProtoDUNE if/when we CAF it?)
SRDetectorMeta pd_hd;    /// Horizontal drift prototype
```

Should parse geometry name?
Or rather use a fcl parameter?

Storing the energy reco information

```
class SRNeutrinoEnergyBranch
{
  private:
    static constexpr float NaN = std::numeric_limits<float>::signaling_NaN();

  public:
    float calo     = NaN;  ///< Calorimetric estimate using all hits
    float lep_calo = NaN;  ///< Lepton (longest track or largest shower) + calorimetric estimate from remaining hits
    float regcnn   = NaN;  ///< Regression CNN (assumes nue hypothesis)
};
```

Do we want to stick with these 3 different energy reconstructions, or add some more granularity to the info? (See H. Souza slides)

- Lepton Ereco method (MCS/Range)
- Different methods to measure the MCS -> (chi2, LogL)

Storing the reco information

```
/// A FD reconstructed neutrino interaction
class SRFDInt
{
  public:
    // these are just placeholders.
    // need to coordinate w/ FD reco folks to put something sensible in here
    std::vector<SRTrack> tracks;
    std::size_t          ntracks  = 0;    // these counters used by SRProxy

    std::vector<SRShower> showers;
    std::size_t           nshowers = 0;
};

/// The information needed to uniquely identify a FD reco object
struct SRFDID
{
  FD_RECO_STACK reco = kUnknownFDReco;  ///< reco stack
  int        ixn  = -1;            ///< interaction ID
  int        idx  = -1;            ///< index in container
};

class SRFD
{
  public:

    std::vector<SRFDInt> pandora;       ///< Reconstructed interactions
    std::size_t npandora = 0;

    /// Convenience function:
    /// Given a specific reco pathway (specified with a SRFD::RECO_STACK value),
    /// an interaction index, and a track index, return the associated reco object
    template <typename T>
    const T & Reco(const SRFDID& id);
};
```

SRFD object to put reco informations for FDs.

What to actually save there?

- All objects as track and shower?
- PFP based on their track/shower likeliness?
- More algos than pandora?

Some more reco info in SRInteraction/SRRecoParticlesBranch

```
class SRRecoParticlesBranch
{
  public:
    int ndlp = 0;  // need these counters for SRProxy
    std::vector<SRRecoParticle> dlp;       ///< Particles reconstructed by DeepLearnPhysics machine learning stack

    int npandora = 0;
    std::vector<SRRecoParticle> pandora;   ///< Particles reconstructed by Pandora

    int npida = 0;
    std::vector<SRRecoParticle> pida;      ///< Particles bearing weights from PIDA algorithm
};
```

What to put there?

PFP/Track/Shower?

Objects to be saved in the CAF file

Currently implemented:

- **cafTree**: StandardRecord tree
- **meta**: Meta tree with pot/run/subrun infos
- **genieEvt**: genie::NtpMCEventRecord for systematics calculation

Anything more needed?

Conclusion

- First version of the updated CAFMaker.
- Still some fields to be filled. Requires some choices to be made.
- Anyone interested/concerned, please take a look at the draft PR (https://github.com/DUNE/duneana/pull/40) and make any comment
- We can discuss here the various topics if you already have some opinions

Waiting for your comments!