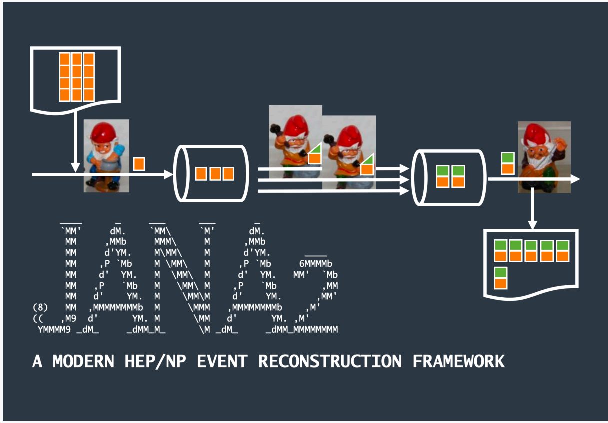
Modular Reconstruction Algorithms Update: Introducing EpicFactory

Nathan Brei
nbrei@jlab.org

ePIC Software & Computing
Weekly Meeting
August 23, 2023



Context

- Sylvester is leading an effort to create *Framework-Agnostic Algorithms*.
- I am involved in that effort, and a lot of this was inspired by talks with him
- This is not that.
- This is an ”80/20” solution focused on improving the user friendliness of the codebase.
- Evolutionary, not revolutionary design
- Opens the door to more exciting things
    - Wiring together and configuring algorithms via an external configuration file
    - Achieving true framework-agnostic algorithms

Starting point

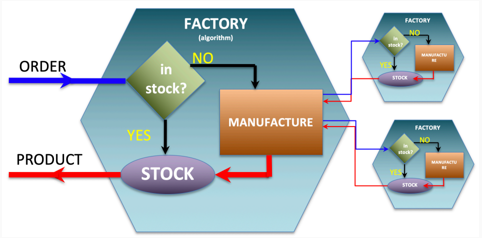
1. Create one base class that combines all the features EICrecon users likely want, *even if they are not presently using them.*
    - Support multiple outputs
    - Support external wiring/configuration like JChainFactory
    - Only use PODIO collections when the output is a PODIO type
    - Consequence: No more confusion regarding when to use JFactoryT, JMultifactory, JChainFactoryT, JChainMultifactory, SetData, SetCollection
2. Leverage "registered members" pattern
    - Already familiar from JANA's JEventProcessorSequential or Gaudi's Properties
    - Declarative paradigm: Allows the framework to do more work behind the scenes
    - Consequence: Simpler, cleaner code
    - Consequence: User doesn't have to reason about the inner workings of the framework

Factories

```cpp
#include <JANA/JFactoryT.h>

class SimpleClusterFactory : public JFactoryT<Cluster> {
    double m_threshold = 1.0;
    std::shared_ptr<BFieldMapSvc> m_bfieldmap_svc;
    std::shared_ptr<BFieldMap> m_bfieldmap;

public:
    SimpleClusterFactory() {
        SetTag("ECalClusters");
    }
    void Init() {
        // Obtain parameters and services here
        auto app = GetApplication();
        app->SetDefaultParameter("threshold", m_threshold);
        m_bfieldmap_svc = app->GetService<MagFieldMap>();
    }
    void ChangeRun(const std::shared_ptr<const JEvent> &event) {
        /// Obtain calibrations/conditions here
        auto run_nr = event->GetRunNumber();
        m_bfieldmap = m_bfieldmap_svc->GetBFieldForRun(run_nr);
    }
    void Process(const std::shared_ptr<const JEvent> &event) {
        auto hits = event->Get<Hit>("RawECalHits");
        std::vector<Cluster*> clusters = clusterize(hits);
        Set(clusters);
    }
};
```



- Lazy: Factories are only run if their data is requested
- Recursive: Factories can call out to other factories
- Memoized: The result is cached so it's only computed once
- Allowed to be stateful, with limits

Multifactories

```cpp
#include <JANA/JMultifactory.h>

class ClusterFactory : public JMultifactory {

public:
    ClusterFactory() {
        DeclareOutput<Cluster>("ECalClusters");
        DeclareOutput<ClusterAssoc>("ECalClusterAssocs");
    }
    void Init() {
        // ...
    }
    void ChangeRun(const std::shared_ptr<const JEvent> &event) {
        // ...
    }
    void Process(const std::shared_ptr<const JEvent> &event) {
        auto hits = event->Get<Hit>("RawECalHits");
        auto [clusters, assocs] = clusterize(hits);
        SetData("ECalClusters", clusters);
        SetData("ECalClusterAssocs", assocs);
    }
};
```

- New as of JANA2 v2.1.0!
- Use this when you have multiple outputs originating from one algorithm

EpicFactories

```
class CalorimeterClusterRecoCoG_factory : public EpicFactory< CalorimeterClusterRecoCoG_factory,
                                                               CalorimeterClusterRecoCoGConfig > {

    PodioInput<edm4eic::ProtoCluster> m_proto_input {this};
    PodioInput<edm4hep::SimCalorimeterHit> m_mchits_input {this};

    PodioOutput<edm4eic::Cluster> m_cluster_output {this};
    PodioOutput<edm4eic::MCRecoClusterParticleAssociation> m_assoc_output {this};

    ParameterRef<double> m_samplingFraction {this, "samplingFraction", config().sampFrac};
    ParameterRef<double> m_logWeightBase {this, "logWeightBase", config().logWeightBase};
    // ...

    Service<JDD4hep_service> m_geoSvc {this};

    // Resource<JDD4hep_service, const dd4hep::Detector*> m_detector {this,
    //        [](std::shared_ptr<JDD4hep_service> s, int64_t run_nr) { return s->detector(run_nr); }}
    // Note: DD4hep detector does not actually appear to be keyed off of run number

    eicrecon::CalorimeterClusterRecoCoG m_algo;
```

EpicFactories

```
public:
    void Configure() {
        // Parameters, services, and logger are automatically set before Configure() is called
        logger()->info("Configured calo clustering with energyWeight={}", config().energyWeight);

        m_algo.applyConfig(config());
        m_algo.init(m_geoSvc().detector(), logger());
    }

    void ChangeRun(int64_t run_number) {
        // Resources and calibrations are automatically fetched before ChangeRun() is called
    }

    void Process(int64_t run_number, uint64_t event_number) {
        logger()->debug("Calo clustering: Processing event# {}", event_number);

        // Inputs are automatically fetched before Process() is called
        std::tie(m_cluster_output(), m_assoc_output()) = m_algo.process(m_proto_input(), m_mchits_input());
        // Outputs are automatically pushed after Process() is called
    }
};
```

Fun things we now get for free

- Reliably reporting factory's configuration parameters at startup
- Reliably reporting factory's input collections at startup
- Reliably updating calibrations and resources when the run number changes
- Reliably configuring a factory-specific logger
- Reliably prefixing parameter names with plugin name and factory tag
- Creating multiple factory instances for the same algorithm without duplicating code

External wiring

- Provides exact same FactoryGenerator interface as JChainFactory
- Also provides a FactoryGenerator interface where parameters are *map<string,string>*
- JWiringService that reads a TOML file already prototyped, but not in this PR

```
app->Add(new EpicFactoryGeneratorT<CalorimeterClusterRecoCoG_factory>(
                 "B0ECalClusters",
                 {"B0ECalIslandProtoClusters",   // edm4eic::ProtoClusterCollection
                  "B0ECalHits"},                 // edm4hep::SimCalorimeterHitCollection
                 {"B0ECalClusters",              // edm4eic::Cluster
                   "B0ECalClusterAssociations"}, // edm4eic::MCRecoClusterParticleAssociation
                 {
                     .energyWeight = "log",
                     .moduleDimZName = "",
                     .sampFrac = 1.0,
                     .logWeightBase = 3.6,
                     .depthCorrection = 0.0,
                     .enableEtaBounds = false
                 },
                 app));
```

Framework independence

- Personal belief: Framework independence is a good goal for the sake of clean design, but a bad goal for the sake of delivering code on time
- We keep an "escape hatch" to access JANA via ***`GetApplication()`***, ***`GetEvent()`***. This way everyone can migrate over without having to do a deep rewrite of their code.
- Currently, in EpicFactory, all other JANA dependencies are hidden behind the EpicFactory base class. This could in theory be swapped with a different base class that provides the same ***`Parameter`***, ***`Input`***, etc, helpers, but uses Gaudi behind the scenes instead.
- To express this formally we would need to invest heavily in C++20 Concepts
- The current implementation uses CRTP. It is easy to get rid of the CRTP. It is much harder to make the algorithm not be a template.

Jefferson Lab

Thank you! That is all.

- User-centric design requires user feedback!
- Check out the pull request at `https://github.com/eic/EICrecon/pull/865`
- Leave a comment
- Or message me on Mattermost!