Jefferson Lab

External Wiring of Factories in JANA2 and EICrecon

Nathan Brei
nbrei@jlab.org

ePIC Reconstruction Group meeting
January 6, 2025

What exists right now?

JOmniFactory

- Interface for using EICrecon algorithms inside JANA2
- Extends JMultifactory
- Provides EICrecon-specific logger
- Provides optional ConfigT structure
- Uses the curiously recurring template pattern
- Wiring is set by the JOmniFactoryGenerator

```c++
struct ClusterConfig {
    double offset = 0;
}

struct ClusterFac: public JOmniFactory<ClusterFac, ClusterConfig> {

    PodioInput<Cluster> m_protoclusters_in {this};

    PodioOutput<Cluster> m_clusters_out {this};

    ParameterRef<double> m_offset {this, "offset", config().offset};

    void Configure() { }

    void ChangeRun(int32_t run_nr) { }

    void Execute(int32_t run_nr, uint64_t evt_nr) {
        for (auto proto : *m_protoclusters_in()) {
            auto cluster = m_clusters_out()->create();
        }
    }
};
```

JOmniFactoryGenerator

- Holds wiring information for each instance of an JOmniFactory
- Parameter values are type-safe thanks to ConfigT
- Allows us to maintain a small number of factory classes
- Parameters, input, and output tags can be overridden on the command line

```c++
extern "C"
void InitPlugin(JApplication *app) {
    InitJANAPlugin(app);

    auto cluster_gen = new JOmniFactoryGeneratorT<ClusterFac>(
                               "clusterizer",     // prefix
                               {"protoclusters"}, // inputs
                               {"clusters"},      // outputs
                               {.offset=1000});   // configs

    app->Add(cluster_gen);
}
```

```bash
# Override this wiring using the following parameters:
# - clusterizer:InputTags
# - clusterizer:OutputTags
# - cluserizer:offset

eicrecon -Preco:clusterizer:InputTags="smeared_protoclusters" input.root
```

What are the problems with this?

JOmniFactory

- No reason for external wiring to be ePIC-specific
- Why are OmniFactories the only kind of factories we want to wire?
- Why are there so many JANA2 factory classes?
- CRTP is inconsistent with the other JANA2 components, forces all callbacks to be implemented

```c++
struct ClusterConfig {
    double offset = 0;
}

struct ClusterFac: public JOmniFactory<ClusterFac, ClusterConfig> {

    PodioInput<Cluster> m_protoclusters_in {this};

    PodioOutput<Cluster> m_clusters_out {this};

    ParameterRef<double> m_offset {this, "offset", config().offset};

    void Configure() { }

    void ChangeRun(int32_t run_nr) { }

    void Execute(int32_t run_nr, uint64_t evt_nr) {
        for (auto proto : *m_protoclusters_in()) {
            auto cluster = m_clusters_out()->create();
        }
    }
};
```

JOmniFactoryGenerator

- The wiring code looks an awful lot like data, except the user is forced to recompile EICrecon to change it.
- Overriding wirings can be done on the command line, but adding or removing wirings can't.
- JOmniFactory cannot be used with basic JFactoryGenerator due to missing defaults for input and output tags.

```c++
extern "C"
void InitPlugin(JApplication *app) {
    InitJANAPlugin(app);

    auto cluster_gen = new JOmniFactoryGeneratorT<ClusterFac>(
                               "clusterizer",     // prefix
                               {"protoclusters"}, // inputs
                               {"clusters"},      // outputs
                               {.offset=1000});   // configs

    app->Add(cluster_gen);
}
```

```bash
# Override this wiring using the following parameters:
# - clusterizer:InputTags
# - clusterizer:OutputTags
# - cluserizer:offset

eicrecon -Preco:clusterizer:InputTags="smeared_protoclusters" input.root
```

Tactical improvements for EICrecon

JWiredFactoryGenerator

- Leave JOmniFactory exactly the same (at least for now)
- Introduce JWiringService and JWiredFactoryGenerator
- The generator will create factory instances for each wiring listed in the wiring file that belongs to the current plugin
- Users can still use JOmniFactoryGenerator for other factories as long as the same wirings aren't specified both places

```c++
extern "C"
void InitPlugin(JApplication *app) {
    InitJANAPlugin(app);

    app->Add(new JWiredFactoryGeneratorT<ClusterFac>());
}
```

```toml
[[factory]]
plugin_name = "clustering"
type_name = "ClusterFac"
prefix = "simple_clusterizer"
input_names = ["protoclusters"]
output_names = ["simple_clusters"]

    [factory.configs]
    offset = "22"

[[factory]]
plugin_name = "clustering"
type_name = "ClusterFac"
prefix = "weird_clusterizer"
input_names = ["protoclusters"]
output_names = ["weird_clusters"]

    [factory.configs]
    offset = "100"
```

Completed

- Implement and test JWiringService
- Implement and test JWiredFactoryGenerator

Todos

- Cut new release and start experimenting with using this in EICrecon
- TOML wiring file should be able to include additional TOML wiring files, in order to factor out different pieces of configuration
- Decide how to store and manage multiple wiring configurations
- (JOmniFactory::Podio{Input,Output} constructors should accept default tags)

Strategic improvements for JANA2

Refactoring

- Reorganize class hierarchy: JFactory takes on JMultifactory, wiring functionality
- Introduce JDataBundle to separate storage from creation logic
- Create WiredGenerators for configuring JEventSource, JEventProcessor, JEventUnfolder, JEventFolder the same way as JFactory

```c++
class ClusterFac: public JFactory {

    PodioInput<Cluster> m_protoclusters_in {this, "protoclusters"};

    PodioOutput<Cluster> m_clusters_out {this, "simple_clusters"};

    Parameter<double> m_offset {this, "offset", 1.0, "Offset in cm"};

public:

    void Process(const JEventKey& event) override {

        for (auto proto : *m_protoclusters_in) {
            auto cluster = m_clusters_out->create();
            cluster.energy(proto.energy());
        }
    }
}
```

```c++
extern "C"
void InitPlugin(JApplication *app) {
    InitJANAPlugin(app);
    // app->Add(new JFactoryGeneratorT<ClusterFac>());     // XOR
    app->Add(new JWiredFactoryGeneratorT<ClusterFac>());
}
```

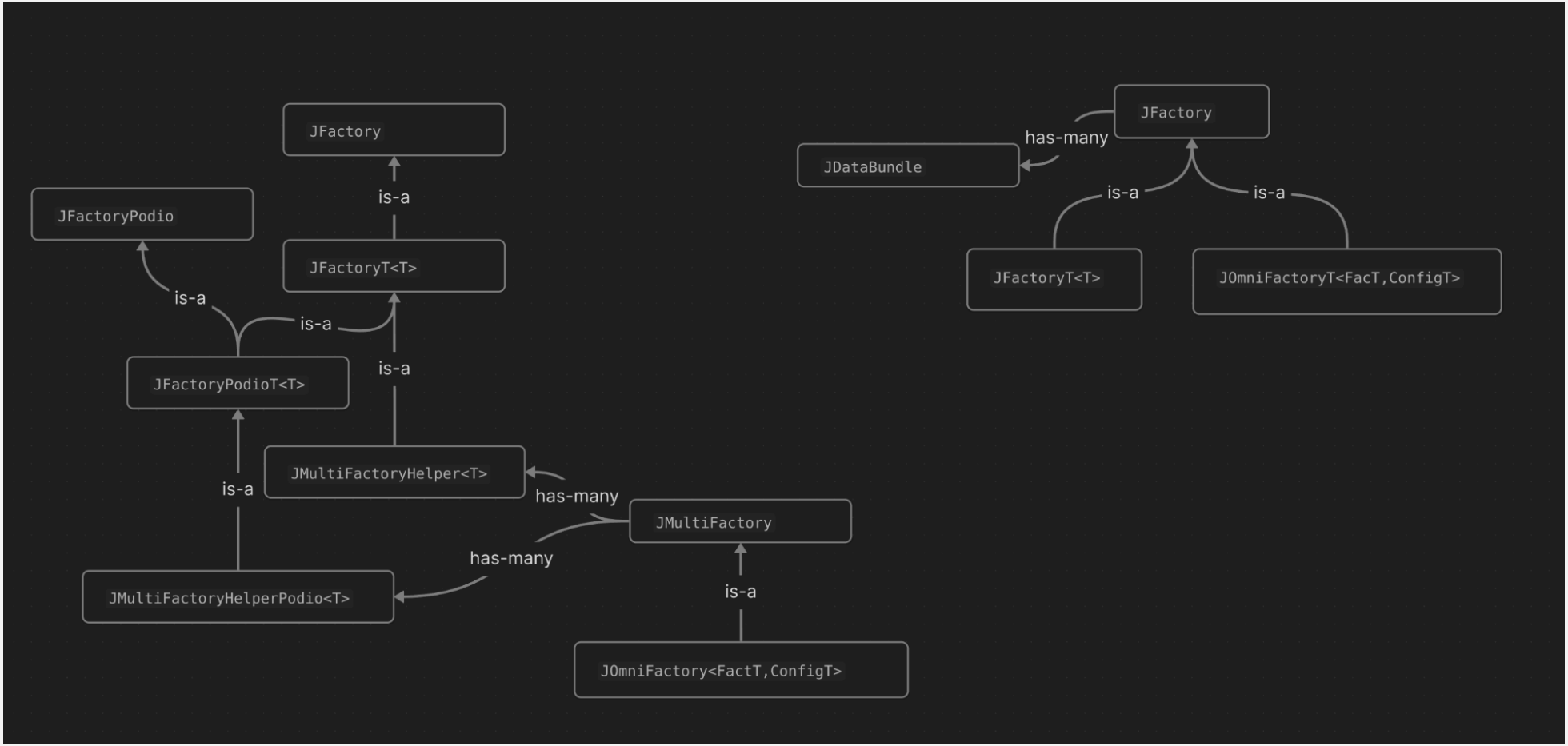

Figure 1: Class diagram for JFactory, before and after

Completed

- Introduce JDataBundle
- Refactor JFactory, JFactorySet, JMultiFactory to use JDataBundle
- Get rid of JFactoryPodioT

Todos

- **Update JANA podio machinery so we can get rid of datamodel glue completely!**
- Refactor JFactoryT use JDataBundle
- Make JMultifactory a synonym for JFactory
- Figure out generalized component generators

Thank you!