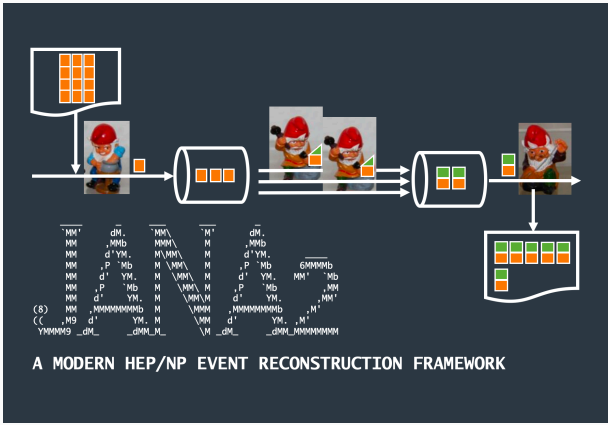
JANA & PODIO Integration: Desiderata and Preliminary Implementation

Nathan Brei
nbrei@jlab.org

ePIC Software Meeting
February 1, 2023



Jefferson Lab
U.S. DEPARTMENT OF ENERGY | Office of Science
JSA

Nathan's personal EICrecon roadmap

1. Factories with multiple outputs (Multifactories)
   - Implemented, currently being tested
   - Can make it into main as early as next week
2. Deep JANA+PODIO integration
   - Designed and partially implemented
   - Full implementation within JANA needs ~~2~~ weeks
   - Full implementation within EICrecon needs another week
   - Will need help with upgrading eicshell/Juggler to the new Podio
   - Will PR some changes to Podio. So far no changes strictly required.
3. SRO with Sergei Furletov
   - Figuring out scope of work now
4. Offloading ML tasks onto heterogeneous hardware
   - Haven't defined scope of work yet
   - Doing some of this work separately for PHASM project

Abstract desiderata

1. If a framework or library is exposed to the user, it must be used idiomatically
   - This applies equally to PODIO and JANA
   - Otherwise, how will users figure out how to use it?
   - Otherwise, how will the integration be robust to upgrades?
   - Otherwise, how will the developers in either camp coordinate?
2. If we add a feature to JANA, it won't conflict with other features in JANA
   - Antipattern: Checking if a feature used by something else is active, and doing something different because of it
   - Violates separation of concerns (Hurts modularity, orthogonality, testability)
   - The user is unlikely to be aware of most JANA features to begin with
   - Open world assumption: JANA code might live in third-party plugins that aren't in the main repo

Concrete desiderata

1. **PODIO object ownership should be left to PODIO collections**
   - Correctness: PODIO collections+frames enforce integrity of object references
   - Performance: PODIO collections allow tight control over the memory layout
   - Usability: PODIO usage needs to be idiomatic (as does JANA usage!)
   - Compatibility: Playing nicely with Key4Hep helps everybody
2. **PODIO objects should still be accessible via normal JANA idioms**
   - JANA+PODIO factories should still track and emit `vector<PodioT*>`
   - However, this should just be a view into the corresponding PODIO collection
   - JANA factories already have a solid mechanism for relinquishing ownership
3. **PODIO collections should be convenient to manipulate in JANA**
   - E.g. `JEvent::GetCollection<PodioT>(string name)`
   - JANA+PODIO factories should support `SetCollection(PodioCollT*)` alongside `Set(vector<PodioT*>)`
   - If possible, calling `Set(vector<PodioT*>)` should still create a PODIO collection under the hood, when appropriate

Implementation: JEventSource_PODIO and JEventProcessor_PODIO

- For each event, all PODIO data is owned by one PODIO frame, which is owned by the JEvent.
- The PODIO event source creates, fills, and inserts one frame into each JEvent, just like the current EICrecon implementation. The user should not need to access this frame directly, although they can.
- The PODIO event processor retrieves and persists the frame for each event.
- The PODIO event source/processor moves from EICrecon into JANA2
- **Make JEventSource_PODIO an abstract class so that we can insert the "datamodel glue" at the EICrecon level. (Ideally, such glue shouldn't be needed at all, however this may require changes to PODIO.)**

Implementation: JEvent and JFactory

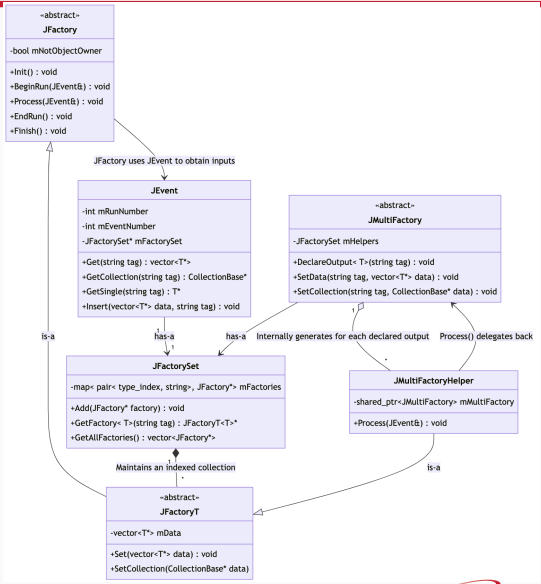
- JANA factories control access to the PODIO frame so that the data is generated on-demand.
- We add `SetCollection<PodioT>()` to JFactory, and update `Set<PodioT>()`. Both should update the frame as well as setting the vector of pointers. (Note that this requires a compile time link between PODIO types and the corresponding collection types. This comes in the form of a temporary generated header file for now, and a simple PR to PODIO in the future.)
- Add `JEvent::GetCollection<PodioT>`. This goes down to the `JFactory` machinery just like `JEvent::Get<PodioT>()`. However, the `JFactory` retrieves the PODIO collection from the PODIO frame as opposed to its own mData cache.

Factories with multiple outputs: Interface

- Factories with multiple outputs inherit from *JMultiFactory* instead of *JFactoryT<PodioT>*.
- *JMultiFactory* exposes the same callbacks as *JFactory* and adds:
  - *void SetData<T>(string tag, vector<T*> data)*
  - *void SetCollection<T>(string tag, CollectionBase* data)*
- *JMultiFactory::Process(...)* calls a setter for each output collection
- MultiFactories need to know their own outputs immediately. (They cannot wait until *Process* gets called). Thus we add *JMultiFactory::DeclareOutput<T>(string tag)* which is called from the constructor.

Factories with multiple outputs: Internal implementation

- *JFactoryT<PodioT>* always produces exactly one collection.
- So does *JFactory*.
- *JMultifactory* generates multiple *JFactoryT*, which delegate back their *Process*.
- The user can add a *JMultifactory* to a *JApplication*, *JFactoryGenerator*, or *JFactorySet* the exact same way they would a *JFactory* and JANA knows what to do with it.



Thank you! That is all.