TOWARDS A FRAMEWORK-INDEPENDENT ALGORITHM LIBRARY FOR EIC AND BEYOND

SYLVESTER JOOSTEN
sjoosten@anl.gov
on behalf of the ePIC collaboration
THE EPIC EXPERIMENT AT THE EIC

We are a large collaboration (160 institutions)
As part of the “Lessons Learned” process, the entire EIC community came together to create a community document to define our aspirations for software and computing for the EIC.

Meant to form a sound foundation to design our software stack.

This document was spread to the entire EIC community through several rounds of open suggestions and endorsement to ensure this is truly a community document.

Endorsed by a large group representing the international EIC community.

100% of responses were positive!

See M. Diefenthaler’s talk from yesterday (Track 5)
https://eic.github.io/activities/principles.html
See D. Lawrence’s talk from yesterday (Track 3)

Software for the Realization of the ePIC Experiment

Our software design is based on lessons learned in the worldwide NP and HEP community and a decision-making process involving the whole community. We are guided by our Software Statement of Principles: https://eic.github.io/activities/principles.html

We will continue to work with the worldwide NP and HEP community.

Modular Simulation, Reconstruction, and Analysis Toolkit using tools from the NP-HEP community

- MC Event Generators
- Detector Simulations in Geant4
- Readout Simulation (Digitization)
- Reconstruction in JANA2
- Physics Analyses

EDM4eic data model based on EDM4hep and podio.
Geometry Description and Detector Interface using DD4hep.

Continuous Integration for Detector and Physics Benchmarks and Reproducibility

We are providing a production-ready software stack throughout the development:
- **Milestone**: Software enabled first large-scale simulation campaign for ePIC.

We have a good foundation to meet the near-term and long-term software needs for ePIC.

slide by Markus Diefenthaler
CHANGING FRAMEWORKS IS EXPENSIVE!

The case for framework-agnostic algorithms

We will leverage heterogeneous computing:
- We will enable distributed workflows on the computing resources of the worldwide EIC community, leveraging not only HTC but also HPC systems.
- EIC software should be able to run on as many systems as possible, while supporting specific system characteristics, e.g., accelerators such as GPUs, where beneficial.
  - We will have a modular software design with structures robust against changes in the computing environment so that changes in underlying code can be handled without an entire overhaul of the structure.

We will aim for user-centered design:
- We will enable scientists of all levels worldwide to actively participate in the science program of the EIC, keeping the barriers low for smaller teams.
- EIC software will run on the systems used by the community, easily.
  - We aim for a modular development paradigm for algorithms and tools without the need for users to interface with the entire software environment.

- Need more rigorous separation of different domains:
  - Framework
  - Algorithms
  - Configuration
  - Resources
  - User workflow
  - ...

- This will enhance user experience, improve maintainability, increase flexibility against future changes, reduce scope of developer responsibility (everyone is the ruler of their own realm)
FROM A USER’S PERSPECTIVE

Close tie-in of algorithms within their frameworks creates friction

- Need to support workflows actually needed by the Users
- Create, test, and run a new reconstruction algorithm with minimal work, support new stand-alone plugins with minimal friction
- Evaluate changes in geometry by changing only the geometry definition and relevant configuration file (no need to change/recompile everything) - again, minimize friction
- Get reproducible (and easily altered) reconstruction configurations without needing to do any additional work (zero-friction reproducibility)
- Provide domains of responsibility where Users of all experience levels can make meaningful contributions
- Distinct domains of responsibility also make clear who to talk to, no more single persons supporting everything at once.
EVOLVING OF THE EPIC RECONSTRUCTION STACK DESIGN

- Strictly modular approach reduces scope of each component
- Easier to onboard new users in any singular piece of the stack
- Every user can find their place based on experience and needs
- Better maintainability and more resilient against changing software needs
- Baked-in reproducibility by enforcing configuration files in every workflow
### DESIGN GOALS AND CHALLENGES

#### Towards a first prototype for algorithms

<table>
<thead>
<tr>
<th>DESIGN GOALS</th>
<th>CHALLENGES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable algorithm sharing across experiments and even communities</td>
<td>Data store interactions</td>
</tr>
<tr>
<td>Framework agnostic algorithms</td>
<td>Properties</td>
</tr>
<tr>
<td>Main dependencies: EDM4hep/EDM4eic and DD4hep</td>
<td>Service integration</td>
</tr>
<tr>
<td>Showcase independence through both Gaudi and JANA2 integration</td>
<td>Context</td>
</tr>
<tr>
<td>No duplication of definitions</td>
<td>“Zero-line” generic framework integration non-trivial</td>
</tr>
<tr>
<td>“Zero-line” holistic framework integration</td>
<td>Automatic testing in a no-framework context</td>
</tr>
</tbody>
</table>
CURRENTLY IN API DESIGN PHASE
Framework integration requires predictable API

Preliminary state machine design; rigorous state validation desired.
EXAMPLE SERVICE INTEGRATION

Towards a first prototype for algorithms

- Services as lazy-evaluated singletons
- Support standalone minimal interface
  - Interface has usable defaults for standalone operation
  - Standalone defaults are meant to be overridden by the framework
- Prototype currently implements LogSvc and GeoSvc
- Special ServiceSvc provides framework with all required services, so it can handle the bindings

```cpp
// Thread-safe lazy-evaluated minimal service system
// C++ Base class to add the instance method
// This could have been part of DEFINE_SERVICE macro, but I think it is better
// to keep the macro magic to a minimum to maximize transparency
template <class SvType> class Service : public PropertyMixin, public NameMixin {
    public:
        static SvType& instance() {
            // This is guaranteed to be thread-safe from C++11 onwards.
            static SvType svc;
            return svc;
        }
        // constructor for the service base class registers the service, except
        // for the ServiceSvc which is its own thing (avoid circularity)
    Service(std::string_view name) : NameMixin(name) { ServiceSvc::instance().add(name, this); }
};
// namespace algorithms

// Note: the log action is responsible for dealing with concurrent calls
// Use default log action as a thread-safe example
class LogSvc : public Service<LogSvc> {
    public:
        using logAction = std::function<void(LogLevel, std::string_view, std::string_view)>;
    void defaultLevel(const LogLevel l) { m_level = l; } // Log level default set
        void defaultAction() const { return m_level; } // Log action default set
        void report(const LogLevel l, std::string_view caller, std::string_view msg) const {
            m_actions(l, caller, msg);
        }
    private:
        Property<LogLevel> m_level("default level", LogLevel::Info);
        LogAction m_action = [] { (const LogLevel l, std::string_view caller, std::string_view msg) {
            static std::mutex m;
            std::lock_guard<std::mutex> lock(m);
            fprintf(stdout, "%s (%s) [%s]", msg.c_str(), caller.c_str(), to_string(l).c_str());
        });
    };
    // ALGORITHMS DEFINE_SERVICE(LogSvc)
};
```
Needed to choose between (1) providing algorithms with a framework allocator, (2) going with a purely functional approach, or (3) passing pointers to already existing objects.

Chose (3) (tuple of pointers) as it significantly simplifies interactions with the frameworks.

Algorithm definition takes an Input and an Output type to define the signature of the ::process function.

Special cases for std::vector<T> (to handle multiple objects of the same type) and std::optional<T> (to handle optional data, e.g. MC truth info in reconstruction algorithms).
OUTLOOK
Towards a first prototype for algorithms

- ✔ Library infrastructure code ready
- 🚧 Framework-agnostic API evaluation
- 🚧 JANA2 bindings in the design phase
- ✗ Algorithm migration within the ePICS stack will follow a successful prototyping phase
  - Seeking “radical modularity" to minimize user friction
- 🚧 Explore collaboration with Key4hep
- 🚧 Gaudi bindings being tested