EIC Software Overview

David Lawrence\textsuperscript{1,*}, Wouter Deconinck\textsuperscript{2,**}, and ePIC Collaboration

\textsuperscript{1}Thomas Jefferson National Accelerator Facility

Abstract. Development of the EIC project detector "ePIC" is now well underway and this includes the "single software stack" used for simulation and reconstruction. The stack combines several non-experiment-specific packages including ACTS, DD4hep, JANA2, and PODIO. The software stack aims to be forward looking in the era of AI/ML and heterogeneous hardware. A formal decision making process was implemented to choose the components that involved everyone in the collaboration that was interested. This talk will present an overview of the software stack currently used for development of the ePIC detector and on which we expect to execute the experiment.

1 Introduction

The ePIC detector is currently under development as the first detector of the new Electron Ion Collider (EIC) facility being constructed at Brookhaven National Laboratory. The detector is designed to measure energetic collisions of electrons and ions (including protons) with center of mass energies in the range of $20\text{GeV} - 140\text{GeV}$ and luminosities up to $10^{34}\text{cm}^{-2}\text{s}^{-1}$\textsuperscript{[1]}. Figure\textsuperscript{[1]} shows a diagram of the basic design of the central detector (the far forward and backward detectors are not shown). With several subsystems and a few $\times 10^k$ channels, the detector is expected to produce data at rates of $O(100)$Gbps out of the counting house\textsuperscript{[2]}. Software is a critical component for integrating simulation, reconstruction, and analysis, all of which are required to complete and understand the detector design details and to execute the experiment when it begins data taking in the early 2030’s. Choosing the components of the software stack has been done to ensure long term support, utilization of modern technologies, and flexibility to incorporate new technologies in the near future.

The ePIC software stack relies primarily on the C++ and Python programming languages. Currently, the C++17 standard is used and compilation assured using the gcc compiler family. A transition to the C++20 standard is expected soon with plans to follow new standards throughout the lifetime of the experiment as warranted. Table\textsuperscript{[1]} lists the core packages comprising the ePIC software stack, some of which are highlighted in later sections.

* e-mail: davidl@jlab.org
** e-mail: wouter.deconinck@umanitoba.ca
2 Design Philosophy

Early in the decision making process of selecting the components of what would become the ePIC software stack, the EIC community developed a Software Statement of Principles\textsuperscript{3}. The purpose was to provide a reference which could be used to help guide the decision making process. This proved to be particularly useful for certain packages where the pros and cons of each of the choices did not make for a clear preference among the EIC software community. An important theme contained in the Software Statement of Principles was modularity. This is considered critical for a software stack that will be used for the next 20 years. The goal of a modular software stack is to allow for new components and technologies to be
adopted as they arise without having to redevelop other components. Modularity is implemented at multiple levels within the stack. The following diagram illustrates the top-level.

![Diagram of Modular Simulation, Reconstruction, and Analysis Toolkit using tools from the NP-HEP community]

3 Containerization and CI

The computing plan for ePIC anticipates utilizing varied distributed compute resources including the Open Science Grid (OSG)\(^4\)[5], the SDCC at BNL, and the SciComp farm at JLab. The OSG alone includes a wide variety of operating systems so to impose uniformity, containerization is used. Singularity/Apptainer is supported in most of the available compute resources and so is the preferred system used by ePIC. Docker images for Intel and ARM architectures are also available for users running on local non-Linux resources such as Windows and MacOS. Singularity images are distributed via cvmfs (see [cvmfs/singularity.opensciencegrid.org/eicweb](https://cvmfs/singularity.opensciencegrid.org/eicweb)). They contain not only binaries of all of the packages in the software stack, but a C++ compiler, Python interpreter, and the development tools (e.g. `cmake`, `git`, etc...) needed to develop software for ePIC. The containers are thus the primary tool for development and production. They are also used for the Continuous Integration (CI) system used to ensure software entering the repositories passes a series of integration tests before being merged into the main branch. Linux users are typically able to retrieve a container and run the software locally on their laptop or PC with a small set of commands (e.g. `curl - -location https://get.epic-eic.org | bash`).

The CI system uses GitHub runners to automatically run a series of integration and other tests periodically and upon every Pull Request (PR). On GitHub, we use a cvmfs and singularity action to load the environment for each job step (using caching for minimal overhead). After an initial set of compilation checks and unit tests, GitHub triggers more extensive benchmark suites on a self-hosted GitLab instance at Argonne National Lab, where 256 cores stand ready to run longer simulations of multiple detector configurations and reference physics processes. Completed benchmarks are reported back to the PRs on GitHub, where they can be consulted for review.

4 The ePIC Software Stack

4.1 Detector Geometry

For the detector geometry the `DD4hep` package\(^6\) was chosen. This uses an XML format to specify the geometry that can be used for both the simulation (via GEANT4) and the reconstruction. The schema includes support for specifying a `readout` type for each detector that a generic GEANT4 program called `ddsim` can use to identify a C++ plugin to implement the readout for the detector. For the purposes of the EIC, an augmented version of this program called `npsim`\(^7\) is used which allows tuning the physics as well as additional levels of customization over the generic `ddsim`. The geometry is organized in a set of directories...
containing the files for a particular group of detectors while the files themselves correspond to specific detectors or components. The system also supports specifying views of the detector that can be used for visualization in yaml files. Several of these are maintained in the same repository as the geometry and readout plugin code.

Figure 2. ePIC detector including the far forward and backward regions as rendered from the DD4hep geometry.)

Figure 2 shows an example rendering of the DD4hep geometry using the ROOT TGeo system. (DD4hep relies on TGeo.) This includes the central detector and the far forward and far backward detector regions. The ePIC detector will be placed at the “6 o’clock” interaction point position or IP6.

4.2 Data Model and I/O

The data model used for storing simulated and reconstructed data is defined in a set of yaml files compatible with the PODIO package. PODIO provides tools to generate C++ code from the yaml source that provide data structures and memory management routines that can be used to manage the objects in memory, write them to a ROOT file, and read them from a ROOT file. The ePIC data model is comprised of a base model (EDM4hep) and an augmentation (EDM4eic). The EDM4hep model is generic and intended to be non-experiment specific. It is maintained outside of the ePIC collaboration, but ePIC members do contribute to it regularly. The EDM4eic data model is written and maintained by the ePIC collaboration. It augments the EDM4hep data model, but allows for faster implementation into tagged releases.

Figure 3 shows a snippet of the EDM4eic data model yaml file. PODIO encourages flat data structures consisting mainly of “plain ’ol data” types such as floats and integers. This makes the ROOT files it produces easy to work with for people familiar with ROOT. Support for specifying one-to-many and many-to-many relations is also present in the PODIO schema.

4.3 Reconstruction Framework

The reconstruction framework used in the ePIC software stack is JANA2, a multi-threaded framework designed for particle physics experiments in NP and HEP. The JANA framework has been successfully used in the past for processing raw data at large HTC/HPC facilities. JANA2 also has new features specifically for supporting streaming systems. Active support for the project at Jefferson Lab, a partnering institution for the EIC, motivated its selection for ePIC.
Figure 3. Example code from the EDM4eic data model yaml file. (See https://github.com/eic/EDM4eic/blob/main/edm4eic.yaml for full source.)

Figure 4. Augmented dependency graph of the reconstruction chain for a single detector in JANA2 framework (see text for details).

Figure 4 shows an augmented dependency graph for the reconstruction chain of a single detector. In this case the Electromagnetic Endcap Calorimeter for the electron-going direction. The cyan colored oval at the top represents the top-level processor object that initially requests the fully reconstructed objects. The blue rectangles represent algorithms required for various stages of the reconstruction. The green trapezoid at the bottom represents objects obtained from the event source (e.g. data file). The yellow rectangle represents associations objects that are also created by the clusters factory. In this graph, the requests for objects flows from the top to the bottom while the reconstructed data objects flow from the bottom to the top in response.
5 Detector Design

The software stack has been useful in helping to finalize design decisions between different detector technologies considered for ePIC. Even without a fully developed reconstruction chain available, the DD4hep and PODIO packages can be used to simulate different detector designs and geometries for comparison. Figure 5 shows an example of this where two candidate technologies for the EM barrel calorimeter where studied to determine particle identification capabilities as a function of momentum in different pseudo-rapidity regions. The PODIO output from the simulation was in the form of standard ROOT files that made them easily accessible for ad-hoc analyses.

6 Summary

The ePIC collaboration at the EIC has selected a software stack based on a guiding set of software principles. The stack includes the general use packages DD4hep, PODIO, and JANA2 among others. Containerization is used both for a full featured CI chain and for software development by collaborators. The software ecosystem is designed to meet the needs of the EIC which is expected to begin data taking in the early 2030s.

Acknowledgement

This material is based upon work supported, in part, by the U.S. Department of Energy, Office of Science, Office of Nuclear Physics under contract DE-AC05-06OR23177.
References


[15] Lawrence, David, Offsite Data Processing for the GlueX Experiment (2020), https://doi.org/10.1051/epjconf/202024507037
