Report for CREMLINplus – 18 months

A well-defined software stack for future HEP experiments is a critical component for simulation, reconstruction algorithms and analysis. The Super Charm-Tau (SCT) factory in the Budker Institute of Nuclear Physics is building a software stack that includes detector simulation, event generators, event data model (EDM), reconstruction algorithms and other relevant software components for this experiment, the Aurora framework. It uses commonly used software tools in HEP: it is based on Gaudi, uses ROOT, Geant4 and integrates components from DD4hep.

These characteristics from the Aurora framework are well aligned with the design and goals of the Key4hep project. Key4hep aims to provide a “Turnkey software stack” with all the necessary ingredients for future experiments. It receives contributions from FCC, CLIC, CEPC and ILD. Synergies between these future experiments allows for the development of the common software stack that includes the best software components from the HEP community. Key4hep is a good fit for SCT software to benefit, contribute and reuse framework stack components being built for the already mentioned large experiments.

The Key4hep project also encourages the use of common tools and software practices to improve on the consistency, maintainability and resiliency of the source code used to build it. Templates are provided to build new packages following common structures, common build systems are agreed along with testing frameworks for the main languages used in Key4hep: C++ and Python. Consistent source code practices are encouraged through tools like clang-tidy and clang-format. A common set of packaging and distribution tools is set with Spack and CVMFS which are widely used.

The existing CLIC simulation and reconstruction software -built for iLCSoft- is in the process of being adapted to run with the Gaudi framework. It was originally built with Marlin and DD4hep, using LCIO for the EDM. The Key4hep project decided to use a different set of tools: Gaudi as the underlying framework and EDM4hep for the EDM. This brings the benefit of using a more mature processing framework, which enables for better use of parallel architectures and future heterogeneous resources. It also benefits from having a larger user base to these tools.

To bring these simulation and reconstruction software to Gaudi the component k4MarlinWrapper is developed. K4MarlinWrapper brings the necessary interfaces, adapters and converters to run Marlin processors with Gaudi. Both frameworks use the same main programming language (C++), but different configuration languages: XML vs Python. A converter between the XML format used by Marlin to the Python format used by Gaudi is developed and improved. It introduces new features to support the concept of constants used in the XML, which enables the propagation of these constant through the configuration file at runtime. The optional Gaudi algorithms are left as commented or deactivated in the conversion; it is up to the user to activate the specific Gaudi algorithms that are marked as optional in the original XML file.

The EDM used by Marlin (LCIO) is not compatible with PODIO based EDMs like EDM4hep or the EDM used by SCT. To interface between these EDMs a set of in-memory converters are developed for k4MarlinWrapper: these enable EDM4hep to LCIO, and LCIO to EDM4hep conversion of user-configured collections, and for user-configured algorithms. The conversion is done in-memory: no intermediate files or persistent format is saved directly from these conversions. By implementing this converters, the original source code of Marlin processors is kept intact thus ensuring the correctness and robustness of its algorithms. Reading and writing events in both EDM formats is also supported by k4MarlinWrapper, it provides mechanisms that allow to use LCIO or EDM4hep input files to read events. To write out persistent events or collections, both the converters and mechanisms for writing are provided for both LCIO and EDM4hep. While the LCIO based writer uses a wrapped Marlin processor, a PODIO-based tool is used to write EDM4hep collections.

The current limitations for the integration of these tools are related to common standards and version of some mentioned packages. SCT uses a PODIO based EDM, similar to EDM4hep but with some differences that make it impossible to use the current converters. New converters or an adaptation of SCT EDM to EDM4hep would be necessary to integrate. While both SCT and Key4hep use the same underlying framework, Gaudi, different versions of it are used, with Key4hep following the latest Gaudi version which introduces some breaking changes in the build system that make it difficult to directly integrate Key4hep components into Aurora.

The geometry integration for the TPC to make simulations while benefiting from iLCSoft packages can be done by adapting the geometry definitions. A geometry definition for the TPC and beampipe is created for this purpose. This geometry is initially based on CLIC's definitions, but adapted to the measurements and materials used in SCT. Missing constants, material definitions and components are added to the materials mixture file. The geometry integration relies on the lcgeo package, which is available to be used with Aurora. To test the correct behaviour and definition of these geometry files DD4hep is used through ddsim. Direct simulation through ddsim incurs in a Python version problem, with Aurora having Python2 based packages and the required DD4hep components using Python3. A new package for Aurora is developed to simulate with this geometry directly from it, instead of through DD4hep. To achieve this purpose, a wrapper for the TPC Sensitive Detector is created to be able to instantiate it, but different instances are used with the underlying Geant4 which impedes the integration of both. A common standard of the underlying Geant4 is needed to integrate the components by the use of "wrappers" that allow to instantiate them.

The common ground of SCT and Key4hep allow for good integration of efforts and components. The differences encountered with package versions and some implementation details present challenges to be able to fully integrate. By solving these challenges SCT and Key4hep could further integrate and SCT could directly benefit from using all the battle-tested reconstruction and simulation algorithms from CLIC through the modern framework developed by Key4hep for experiments like FCC, CLIC or CEPC.