Project information

Project full title	Connecting Russian and European Measures for Large-scale Research Infrastructures – plus
Project acronym	CREMLINplus
Grant agreement no.	871072
Instrument	Research and Innovation Action (RIA)
Duration	01/02/2020 – 31/01/2024
Website	www.cremlinplus.eu

Deliverable information

Deliverable no.	D5.1 (D44)
Deliverable title	Status report on the software for the SCT detector
Deliverable responsible	CERN
Related Work-Package/Task	WP5; task 5.3
Type (e.g. Report; other)	Report
Author(s)	Andrey Sukharev, Placido Fernandez Declara, Vitaly Vorobyev, Andre Sailer
Dissemination level	Public
Document Version	1
Date	31 Jul 2021
Download page	www.cremlinplus.eu

Document information

Version no.	Date	Author(s)	Comment
1	18/07/2021	A. Sukharev, P. Fernandez Declara	First draft
2	19/07/2021	V. Vorobyev	Generation and Analysis modules
3	21/07/2021	Andre Sailer	Sec. 3 improved

Table of Contents

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 871072.

1. Introduction

A well-defined software stack is an essential component of any High Energy Physics (HEP) experiment. Such a stack (usually referred to as the software framework) should provide means to perform all conventional data handling steps (Fig. 1): Monte Carlo generation of signal and background events, simulation of the detector response, reconstruction for both simulated and recorded experimental events, final physical analysis, etc. The Super Charm-Tau (SCT) factory detector team in the Budker Institute of Nuclear Physics is building a software stack that should cover these tasks, the Aurora framework. The Aurora framework is going to use as much as possible of the existing HEP software, either directly relying on the general-purpose projects like Gaudi, Geant4, ROOT, and Key4hep, or porting and re-using relevant components from other experiments, for instance, general Aurora structure and build system is inspired by Athena of ATLAS, lots of framework components were taken from the FCC software, many analysis tools were adopted from the Belle2 experiment.

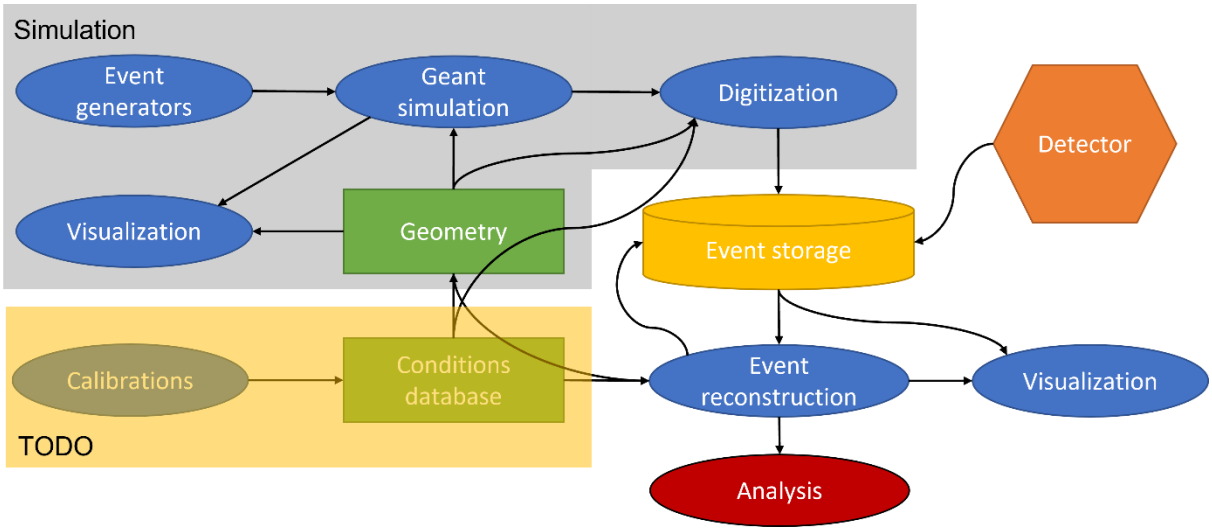

Fig. 1. Main data flows and operations in the detector software.

The software framework is a critical component of the detector project from its very early stages. It should allow physicists to evaluate various options of the detector subsystems and to estimate the resulting detector performance thus outlining the possible physics programme. On the other hand, not all the components shown above are mandatory at the detector design stage, but they must be kept in mind to make further framework development easier.

For joint software development three services are provided: a Git server https://git.inp.nsk.su and a software development server proxima.inp.nsk.su which also contains Aurora installations. A Wiki server has been set-up as a tool for sharing technical information: https://ctd.inp.nsk.su/wiki. While the wiki page is publicly accessible, the two other services are accessible to all members of the SCT team, including CREMLINplus colleagues.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 871072.

2. The SCT software

The SCT detector software framework named Aurora is based on the de-facto HEP standard framework Gaudi. Aurora covers all the aspects of software components interaction at run time, including configuration, data exchange and job running.

The Aurora framework architecture follows the traditional scheme: it separates code and data. Physics data are processed by Algorithms on an event-by-event basis. An algorithm takes input data, manipulates it, and produces new output data. Some parts of this work may be delegated to Tools. The framework provides data stores for data exchange between algorithms. Tools and algorithms are implemented with the C++ programming language. The Python programming language is used for configuration and steering scripts.

The software is organized in packages, or modules, each module is responsible for a set of well-defined tasks. General-purpose, control, service, example and other modules are developed by the software team, while the modules dealing with a certain detector subsystem and with specific physics tasks (like event generation and physics analysis) are expected to be prepared by the corresponding experts. The required set of “external” software (compilers, unmodified standard HEP software libraries and their dependencies) and the build system are also maintained by the software team.

The data flow in the processing chain is typical. Events are generated by the chosen generator, then injected into the simulation. The simulation could be either parameterized or full. Full simulation tracks all the particles in the digital model of the detector, yielding the event picture. Then the picture is processed with digitization modules, their output data format is exactly the same as it would be for the real detector hardware. Thus, the reconstruction could operate with either simulated or real data in the future. Reconstructed events are analysed using the corresponding high-level tools. The parameterized simulation does not feature the detailed description of the detector and just directly converts initially generated particles to the format usable for high-level analysis, the conversion uses the detector parameters like momentum and energy resolution as pre-defined data.

On each stage of the data processing chain the intermediate data could be routed directly from one module to another or stored and read back later.

2.1. Event generators

Interaction of electrons and positrons in the SCT beam collision region leads to production of various particles, most of which have many decay channels. The processes of particles production and decays are simulated with a dedicated software called event generators. The Generation package contains algorithms for event generation.

A GenAlg algorithm is executed for event generation. It defines unified interfaces for all high-level event generators in Aurora. A IHepMCProviderTool interface must be implemented for each high-level event generator. The method getNextEvent of this interface fills a HepMC3 object. Therefore, at

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No. 871072.

the low level, all SCT event generators produce events in the HepMC3 format, that is the de-facto standard in high energy physics. Besides, GenAlg has interfaces for vertex smearing and events merging tools.

The SCT software adopted an event data model based on plain-old-data (POD) objects just as suggested by the Key4hep initiative. A HepMC3 object is not a POD, that is why a HepMC3 to SCT event data model (EDM) converter HepMCToEDMConverter is implemented and is called after the GenAlg. Events in the SCT EDM can be serialized/deserialized or transferred online to the next simulation step.

The SCT EDM is implemented with the PODIO package. The EDM4Hep package, a candidate for the standard EDM by Key4Hep is based on PODIO. That makes natural the planned transition of the SCT EDM to EDM4Hep.

The EvtGen is the main high-level event generator in Aurora. It contains many features that cover significant part of the SCT experiment needs:

- Extendable list of particles and its properties.
- Extendable list of decays of charmed hadrons and tau lepton that reflect current knowledge of these particles.
- Simple user interface to specify processes to generate.
- Convenient developer interface to define new decay amplitudes (together with a large list of high-quality predefined decay amplitudes).

A more precise generator of tau lepton decays is Tauola. Tauola is included in Aurora and invoked through EvtGen.

The effect of final-state-radiation is needed to be considered in precise studies. This effect is simulated with the PHOTOS package also accessed through EvtGen.

There are many hadronic decays that are not described explicitly. A general generator of hadronic decays Pythia helps in such cases. It also included in Aurora and is accessed through EvtGen.

The EvtGen – Tauola – Pythia – PHOTOS combination is the core stack of event generators available in Aurora. It is suitable for generating general sets of events (“all possible processes for a given beams energy”) as well as for generating a specific subset of “signal” processes.

Other generators will be added to Aurora when needed. The recommended way of adding new generators is making new EvtGen models (although this is not always possible, e.g., for the two-photon processes generation). That would provide a uniform user interface for all generators in the system.

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No. 871072.

Several event generator configuration files are added to Aurora as examples to help new users getting use with generator configuration.

A high-level Python interface for event generators is implemented on top of the interface provided by the framework. A user only needs to specify EvtGen as generator, and select the root particle, configuration file and beam energies.

2.2. Detector subsystem geometry

A uniform detector geometry description should be provided for all software modules thus making sure the same geometry is used at all stages of data processing. This is achieved using the DD4hep toolkit. To describe the geometry, one must provide the XML files with general information and the C++ code which interprets the XML data and builds corresponding objects. Later any module could access the geometrical information via these objects. If a user needs to slightly adjust the geometry changing the XML files only, this does not require to re-build the whole program.

Each detector subsystem has a corresponding geometry software package, containing the XML description, the C++ code to interpret the XML, and some extra files like build rules and tests. If there are several options for a subsystem, each option is represented by a separate package. The desired variant is selected at run time, allowing to compare the options.

Each subsystem declares certain areas as the sensitive volumes to produce simulation output data which will be used as input for digitization or reconstruction. A unified SensitiveDetector class allows to do this automatically.

The list of geometry packages includes (developed by Budker INP software team and subsystem experts unless explicitly specified):

1. The FinalFocusGeo and The BeamPipeGeo, representing the beam pipe and rather complicated final focus magnets and cryostats, based on the technical sketches provided by the collider team.

2. Three options for the Vertex Tracker: the TPCGeo – time-projection chamber, the CmuRWELLGeo - cylindrical modular μ-RWELL detector being developed by the INFN group, and the SiStripGeo – simple four-layered Silicon plates device with strip readout.

3. The Drift Chamber geometry package.

4. The FARICHGeo for the PID subsystem, with the upcoming FDIRC option from the JLU Giessen/University of Giessen team.

5. The Calorimeter package describing both barrel and endcap crystal calorimeter, with a special algorithm to calculate exact position and sizes for each crystal.

6. Simplified description of the superconducting magnet in the CoilGeo package. There is also the ThinSolenoidGeo featuring the coil before the calorimeter, however this option is not likely to be chosen.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 871072.

7. The MuonSystemGeo, describing the muon system and the iron yoke of the magnet. There is also the code to provide the magnetic field map.

The overview of detector geometry is shown in Fig. 2.

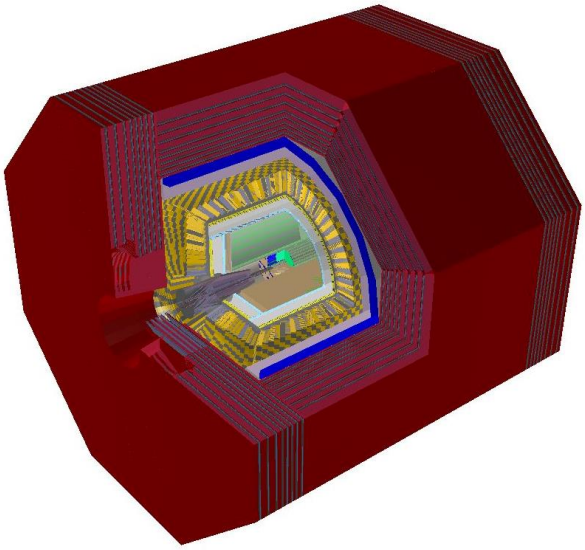

Fig. 2. The detector geometry.

2.3. Signal digitization

The data received from the (future) detector would most likely be organized as a set of pairs of numbers [the ID of electronics channel activated in the event; the value measured by the channel]. On the other hand, the simulation output is generally a set of places in the detector where a particle deposited some energy. The digitization is responsible to convert the simulation output to the real detector data format thus allowing the reconstruction to work both with simulated and real data in a uniform way. The conversion could be complicated, taking into account effects like electron drift, or light absorption.

Ultimately, all detector subsystems should follow the scheme. This is, however, not the case yet, and most subsystems perform reconstruction based on the simulated data directly, which is more or less acceptable at the present stage of the project. The exception is the SiStripDigi module serving as an example. For the Silicon Strip inner tracker option, the digitization procedure is rather simple: the place where the energy is deposited in the tracker determines the electronics channel ID (actually two channels activated for each deposition: longitudinal and transverse ones), while the “measured” value is just the energy. The geometry is addressed via DD4hep objects instantiated by the corresponding geometry package SiStripGeo. Various effects of charge collection are planned to be addressed later.

Based on the example, some subsystem groups are already developing their digitization modules too.

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No. 871072.

2.3. Event reconstruction

The reconstruction is generally a two-stage procedure. Ideally, at the first stage the information received from the simulation or (in future) from the detector is processed to obtain from raw event data – the channel IDs and their readouts – the meaningful physical information. This could be, depending on the subsystem, spacial coordinates and times of a particle flight, energies deposited in certain parts of the detector etc. The second stage combines these data to form high-level reconstructed objects: tracks of charged particles with momenta in the vertex and main trackers, total energy deposits associated with particles in the calorimeter, particle identification indicators and so on. Finally, the reconstruction yields a set of final state particles with their properties to be used as the input for the analysis.

The first-stage reconstruction is prepared for each subsystem by the experts and placed in a corresponding module with -Rec suffix. Implemented at the moment are SiStripRec and MuonSystemRec. The advanced RecCalorimeter module reconstructs clusters of energy deposits in the calorimeter. Detector elements without sensitive volumes do not need reconstruction module.

The second stage reconstruction is implemented at the limited level. Lots of already developed advanced algorithms have not been merged into the master code base yet:

1. The drift chamber track finding and fitting based on the GenFit library.
2. Another tracking solution described below in section 3.
3. FARICH reconstruction code.

2.4. Physics analysis

Physics analysis of events collected by the SCT experiment requires the application of many algorithms and sophisticated techniques of statistical analysis. The first stage of almost any analysis – selection of potentially interesting events – is well standardizable. An almost complete list of actions needed at first stage is following:

- Accessing final-state-particles (FSPs) reconstructed in the detector.
- Combining FSPs in a tree structure to build candidates for decays of unstable intermediate particles.
- Imposing requirements on selected particles (e.g., particle momentum or azimuth angle not less than some value).
- Imposing kinematic assumption on a decay tree (e.g., forcing mass of intermediate particle to a fixed value).
- Calculation of various parameters for a particle candidate or for an event and saving it for further analysis.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 871072.

The high-level interfaces for the listed actions (and some other that are omitted for the sake of simplicity) are implemented in the Analysis package of Aurora. This package is inspired by Belle II software, and we express gratitude to the Belle II collaboration for sharing source code with SCT team. Nevertheless, direct use of Belle II software is not possible since it uses BASFII framework incompatible with Aurora and Gaudi, so most of the code was written from scratch or heavily modified starting from the Belle II original code. All references to the original code are kept in the header commentaries.

The Analysis module contains the following key components:

1. Particle class, that implements many tools needed for SCT data manipulation. Particle objects are nodes in the reconstructed decay trees.
2. ParticleList class – a specialized container that is aware about antiparticle concept.
3. EventLoader – Aurora algorithm that accesses reconstructed FSPs and produces ParticleList objects for further analysis within the Analysis package.
4. DecayDescriptor sub package describing user-friendly language of decays description. The following are examples of correct expressions a user can use to in analysis:
   a. `D0 -> K- pi+`
   b. `D0 -> [rho0 -> pi+ pi-] pi0`
   c. `D0 -> [rho0 -> pi+ pi-] [rho+ -> pi+ [pi0 -> gamma gamma]]`

   (here `D0`, `K-`, `pi+`, `pi-`, `pi0`, `rho0`, `rho+`, and `gamma` are names of predefined particles).
5. ParticleCombiner algorithm generates candidates of decays of intermediate particles (e.g., for the D0 -> pi+ pi- decay, this algorithm loops over all pairs of positive and negative pi mesons and created a D0 candidate for each pair). Consecutive application of the ParticleCombiner algorithm is used to produce arbitrary decay trees.
6. AnaVarManager sub package is extendable library of predefined variables. A variable can be calculated for a given Particle object (e.g., energy, transverse momentum). This package is used when selection criteria are imposed on list of candidates and when a set of variables is calculated to save an n-tuple for further analysis.
7. Kfit sub package implements kinematic fitters. Only mass-constrained fitter is implemented at the moment. Implementing algorithms for other kinds of kinematic fitting is a subject of improvement of the Analysis package.
8. NtupleAlg algorithm takes a list of reconstructed particle candidates and a list of variables to calculate, calculates each variable for each particle and saves the result in a plain ROOT TTree format – an n-tuple.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 871072.

Fig. 3 shows a typical process of data manipulation within the Analysis package. It starts with access to data in the SCT EDM format with EventLoader, continues with a user-defined set of algorithms, and finishes with writing n-tuple with the NtupleAlg algorithm.

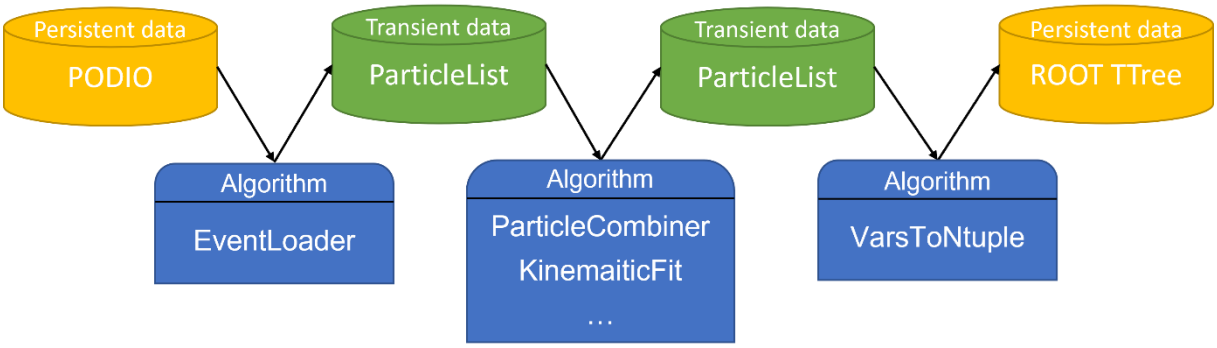

Fig. 3. Scheme of algorithms and data manipulation implemented in the Analysis package.

The Analysis package addresses the following issues:

1. It provides high-level instruments for the data analysis allowing fast learning for newcomers.
2. It minimizes time of writing analysis code.
3. It reduces number of code bugs since a physicist does not need to implement (and debug) various standard algorithms.
4. It ensures reproducibility of the analysis since all algorithms are well-documented and eventually will be well-tested.
5. It standardizes the output format of the analysis since structure of the output n-tuple is generated automatically. This feature allows development of software for further sophisticated analysis assuming the common format of input data.

The Analysis package is supposed to be used in the upcoming SCT physics case studies as well as in analysis of the SCT experiment data.

2.5. Service tools

A set of service tools is available for the SCT software users. The visualization is provided by the GeoDisplay and the EventDisplay tools based on DDEve from the DD4hep toolkit. The first one draws the detector geometry in 3D, and it is robust enough, while the second one aiming to visualize the event data still requires lots of improvements.

The geometry validation package includes tools to check the geometry description for inconsistencies like overlapping volumes (both at the DD4Hep level and in the Geant4 simulation). There are also geometry hierarchy printing and material scanning tools based on DD4Hep.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 871072.

Using these tools, a geometry test mechanism is implemented. The tests should be prepared by a developer for each geometry package, helping in the early detection of errors when modifying the geometry.

The build system of the software stack is based on one from the ATLAS experiment. External standard HEP packages are built using LCGCMake infrastructure. The software development workflow utilizes GitLab as a version control and project management system.

3. Inter-operation with Key4hep & iLCSoftware

The characteristics of 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. Like the Aurora framework, the Key4hep stack is based on Gaudi as the underlying framework and DD4hep for the geometry information. The Key4hep EDM (EDM4hep) is also based on PODIO but currently differs from the SCT-EDM.

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 iLCSoft framework, among other things, contains tools for the simulation and reconstruction, including pattern recognition and trackfitting, of a Time Projection Chamber (TPC). Other iLCSoft components might also turn out to be of use for SCT as well. As a TPC is one of the possible options for the SCT vertex detector using these components from iLCSoft, would allow one to avoid the re-implementation of these tools in the Aurora framework. As part of the Key4hep project, the iLCSoft components are in the process of being adapted to run with the Gaudi framework, instead of the Marlin framework and LCIO EDM of iLCSoft. To bring this iLCSoft based simulation and reconstruction software to Gaudi in a generic way, the component k4MarlinWrapper is developed. The 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 constants through the configuration file at run time. 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.

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No. 871072.

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 these 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 ILD'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, however the TPC sensitive detector needed to be used along with the rest of the iLCSoft based reconstruction tools is not compatible with the Aurora geant4 simulation. A common standard of the underlying Geant4 is needed to integrate the components by the use of "wrappers" that allow to instantiate them. To test the correct behaviour and definition of these geometry files, DD4hep is used through ddsim and the reconstruction of tracks in the SCT TPC was tested using the k4MarlinWrapper

The common ground of SCT and Key4hep allows 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 iLCSoft through the modern framework developed by Key4hep for experiments like FCC, CLIC or CEPC.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 871072.

4. Conclusion

The SCT detector software development has advanced a lot since the beginning of the SCT project. The Aurora release 1.0.1 available since June 2021 comprises the software set minimally required at the present stage of the detector project, including:

1. primary event generators,
2. parameterized and full simulation,
3. detector geometry description (with at least basic description for all detector elements, and several options for some subsystems),
4. sample digitization module,
5. reconstruction modules (from basic to really advanced, depending on subsystem),
6. analysis and job configuration tools,
7. test and service tools.

The development of new modules covering the yet missing functionalities, further Key4hep integration, modernization of the core software components being used by the framework, build and visualization systems improvements are the nearest goals for the SCT software team and the detector experts.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 871072.