Software framework for the Super Charm-Tau factory detector project

Maria Belozyorova[1,2], *Dmitry* Maksimov[1,2], *Georgiy* Razuvaev[1,2], *Andrey* Sukharev[1,2], *Vitaly* Vorobyev[1,2], *Anastasia* Zhadan[1,2,*], and *Daniil* Zhadan[1,2,**]

[1]Budker Institute of Nuclear Physics, 11, akademika Lavrentieva prospect, Novosibirsk, 630090, Russia
[2]Novosibirsk State University, 1, Pirogova street, Novosibirsk, 630090, Russia

Abstract. The project of Super Charm-Tau (SCT) factory — a high-luminosity electron-positron collider for studying charmed hadrons and tau lepton — is proposed by Budker INP. The project implies single collision point equipped with a universal particle detector. The Aurora software framework has been developed for the SCT detector. It is based on trusted and widely used in high energy physics software packages, such as Gaudi, Geant4, and ROOT. At the same time, new ideas and developments are employed, in particular the Aurora project benefits a lot from the turnkey software for future colliders (Key4HEP) initiative. This paper describes the first release of the Aurora framework, summarizes its core technologies, structure and roadmap for the near future.

1 Introduction

Proposed by Budker INP team Super Charm-Tau (SCT) factory[1] is a symmetric electron-positron collider with a single collision point going to operate in the energy range between $\sqrt{s} = 2$ and 6 GeV. This energy range covers the charmonium family (including the $J^{PC} = 1^{--}$ states J/ψ, $\psi(2S)$, $\psi(3770)$ and others that can be produced directly), several open charm hadrons thresholds ($D^{(*)0}\overline{D}^{(*)0}$, $D^{(*)+}D^{(*)-}$, $\Lambda_c\overline{\Lambda}_c$ etc.), and the τ lepton threshold, thus providing the rich physics program. Projected luminosity of the collider is 10^{35} cm^{-2}s^{-1}. Maximal event rate to deliver to the data storage is expected to be more than 350 kHz for operation at the J/ψ resonance.

The collision point is equipped with a universal particle detector covering nearly full solid angle (Figure 1). The detector contains all traditional subsystems:

- *tracking system* composed of the drift chamber and inner tracker. Two main alternative options for the inner tracker are under discussion at the moment: time projection chamber (TPC) and cylindrical gas electron multiplier (cGEM), the backup option being silicon strips,
- *charged particles identification system*, where Cherenkov-based and time-of-flight solutions are under consideration,
- *calorimeter*. Pure CsI crystal calorimeter is the baseline option,

*e-mail: A.A.Zhadan@inp.nsk.su
**e-mail: D.S.Zhadan@inp.nsk.su

- *muon system*. A scintillator counter system similar to the Belle II KLM is considered as a suitable technology,
- *superconducting coil* providing uniform magnetic field of about 1.5 T in the tracker.

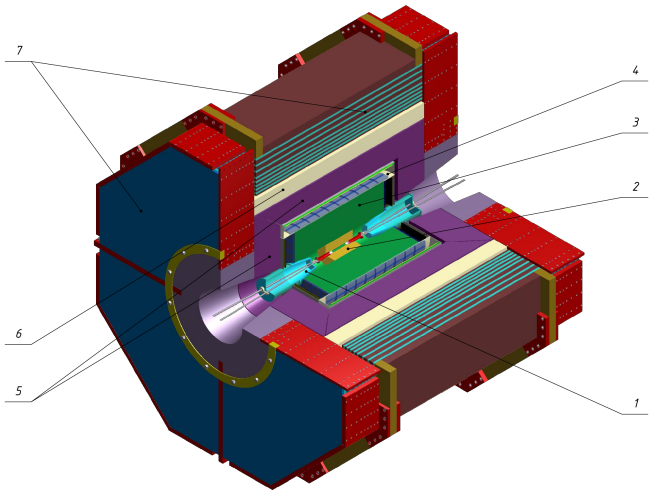

Figure 1. The SCT detector engineering sketch. 1 — Beam pipe and final focus magnets, 2 — Inner tracker, 3 — Drift chamber, 4 — PID system, 5 — Calorimeter, 6 — Magnet coil, 7 — Muon system and yoke.

From the software point of view, the SCT experiment is a typical heavy flavor factory experiment. It requires a complete stack of relevant software, including

- event generators,
- parametric and full detector simulation,
- event reconstruction algorithms,
- online event interpretation for trigger decisions,
- event data model (EDM),
- I/O interface to conditions data base,
- I/O interface to data storage,
- offline data analysis algorithms,
- build system and release management software.

These components and their interconnections are usually referred to as the *software framework*.

Main components of the full framework and its information flows are illustrated in Figure 2. Although not all the components presented there are really mandatory at the detector design stage, we must keep them in mind to make further framework development easier.

We adopt the following guiding principles to work on the detector software:

1. Rely on trusted existing solutions where it is possible.

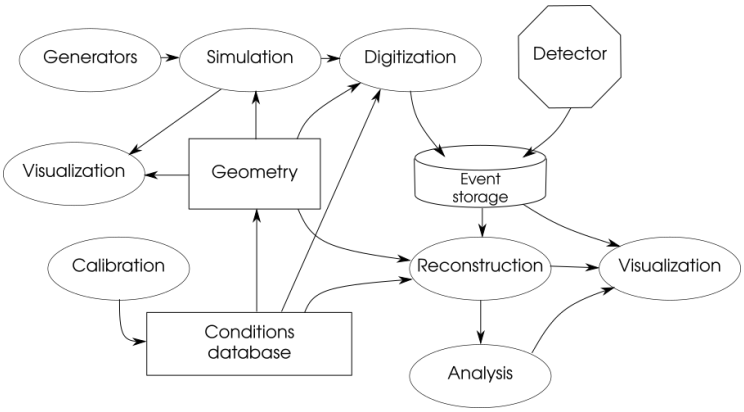

Figure 2. The SCT detector software and data flows scheme.

2. Adopt the best practices of collaborative software development, including the code style guidance and code reviewing procedure.
3. Be in touch with and contribute to the global community of the detector software developers.

In particular, we follow the Key4HEP[2] project and we look forward to benefit from it.

2 The framework

The SCT detector software framework is based on Gaudi[3] and has name Aurora. It covers all the aspects of software components interaction at runtime, including configuration, data exchange and job running. Aurora uses Semantic Versioning[4] to indicate breaking changes in its public API and provided abilities. A breaking change will result in a new major version.

The framework architecture follows the traditional scheme: it separates code and data. Physics data are processed by Algorithms on 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 C++. Python is used for configuration and steering scripts. The standard computing environment for the framework is Scientific Linux 7 at x86_64 architecture, with GCC 9 as the main compiler. Python 2 is still used, however transition to Python 3 is anticipated in the nearest future.

The data flow in the processing chain is conventional. Events are generated by the chosen generator, then injected into the simulation, yielding the event picture in the detector. The picture is processed with digitization modules, their output data format being exactly the same as it would be for the real detector hardware. Thus the reconstruction could operate with either simulated or real (in future) data. Reconstructed events are analyzed using corresponding high-level tools. On each stage of this chain the intermediate data could be routed directly from one module to another, or stored and read back later.

2.1 Event Data Model

The SCT detector event data model (EDM) is implemented as C++ plain-old-data (POD) objects. Data persistence is currently implemented via the ROOT[5] format. This format provides fast and straightforward I/O interface.

The EDM objects are defined in a single yaml[6] file. PODIO[7] package is used to generate actual C++ classes and provides serialization mechanisms to store/retrieve data. Described scheme is embedded in the Aurora framework via the DataModel package. The EDM is not stabilized yet, as the software development process often requires to introduce new data objects.

Large data flow and large volume of stored data are expected during operation of the experiment. It imposes strict requirement on the SCT EDM. It should be carefully designed to provide fast I/O, optimal event size and standardized interface for the data analysis algorithms.

We consider to rebase our data model to the EDM4hep[8] later.

2.2 Event Generators

The required set of event generators should provide a reasonable description of e^+e^- interactions in the energy range between 2 and 6 GeV. Generators can be divided into three categories

1. Exclusive decays of hadrons and tau lepton. The EvtGen[9] generator covers the decays of hadrons, and Tauola[10] generator cover the decays of tau leptons. Final-state-radiation is simulated by the PHOTOS[11] generator. Inclusive fraction of charmonia decays is simulated with the Pythia[12] generator.

2. Inclusive generators for $e^+e^- \to$ hadrons. This is a challenging task and only a preliminary solution based on Pythia generator is implemented at the moment. Experimental input and careful parameter tuning are required to achieve reasonable description of the $e^+e^- \to$ hadrons processes in the energy range of the SCT experiment.

3. Generators for luminosity measurements and calibrations (MCGPJ [13], BabaYaga [14], BBBREM [15], KKMC [16] and other).

Primary generators produce output in the HepMC3[17] format which is immediately converted to SCT EDM and can be serialized or used as input to the SCT detector simulation algorithm.

2.3 Detector geometry description

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. Each detector subsystem has a corresponding geometry software package. If there are several options for a subsystem, each option has a separate package. For example, there are TPCGeo, CmuRWELLGeo and SiStripGeo packages for the inner tracker. A user must select the desired variant when starting a job.

The DD4Hep[18] toolkit is our choice for this task. The geometry description consists of one or more XML files containing subsystem elements shapes, sizes, materials, visualization attributes, readout parameters etc. The corresponding C++ class interprets the XML description and creates all necessary objects.

2.4 Detector response simulation

This is actually a two-stage process. First, the detector simulation module uses Geant4[19] for particle propagation and hit generation. DD4Hep provides a special tool DDG4 to supply

the uniformed geometry description to Geant4. At the end of the simulation stage, each generated hit contains information about the corresponding minimal detector element (called *segmentation cell* in DD4Hep terms and determined by readout parameters set in geometry description) and energy deposited there. This is a so-called `G4Hit`.

Digitization, i. e. conversion of hits generated by Geant4 to a form of data read from real detector hardware (`RawHit`), is performed by ad-hoc algorithms, sometimes quite trivial, sometimes rather complicated, developed by subsystems experts. The `RawHit` consists of an electronics channel identifier and data "read" from the channel.

2.5 Event reconstruction

This is also two-level procedure. Basic reconstruction reads `RawHit` sets for the subsystem and uses them to restore positional/energetic/temporal information forming corresponding reconstructed hits. Each subsystem group should provide a basic reconstruction package for its subsystem.

Then, the reconstructed hits are combined in a subsystem or in several subsystems to form tracks, clusters or some another high-level objects. Among the options for the track reconstruction package GenFit[20] and ACTS[21] are considered at the moment.

Calorimeter reconstruction is based on a generic topo-cluster technique.

There is no final solution yet for Cherenkov light rings reconstruction. Various algorithms including parallel processing at FPGA are under investigation.

2.6 Parametric detector model

A parametric simulation is a tool to receive a detector response without detailed description of interaction of particles with matter. The detector response is obtained using random number generating whose distribution density is described with a small set of parameters. Initially developed as a standalone Python application, the parametric simulation has been recently merged into Aurora and now is being converted to C++ code where it is appropriate. The development and testing goes on continuously to keep the parametric simulation in sync with updates in subsystem-specific and general software.

The main algorithm collects responses from each detector subsystem for each primary generated particle. Responses are stored in an output collection, while input particles can be generated in advance or just in time. Each detector subsystem (there are tracker, calorimeter, FARICH PID and muon subsystem at the moment) is implemented as a Gaudi tool with the corresponding interface. The tracker and the calorimeter smear particle parameters according to the Gaussian distribution, while the FARICH PID and the muon subsystem use for that purpose the results of a pre-conducted standalone full Geant4 simulations. There is also a helper package for geometrical objects and operations like lines and helices and their intersections with cylinders and planes.

The parametric simulation yields the detector response in the SCT EDM format thus allowing to analyze its result in the same manner as the result of the full simulation.

2.7 Offline data analysis

The Analysis module implements high-level tools for the offline analysis of reconstructed events. The analysis procedure is candidate-based meaning that candidates of the process of interest (e. g. $D^0 \to K^-\pi^+$ decay) are searched in each event, and necessary information is saved for each candidate. The guiding principles behind software for the offline analysis are: 1) the analysis tools should implement all required data manipulations, 2) the analysis tools

should be easy to use, 3) an analysis should be easily reproducible, and 4) the analysis result should be in a standardized format. These principles imply that a user would be forced to use only provided high-level tools and not process low-level data him or herself and impose strict requirements to the analysis tool: to be versatile, extendable, and user-friendly.

The following features are implemented in the Analysis module at the moment:

- accessing reconstructed final-state-particles (input for the analysis),
- formation of candidate list for arbitrary particle decay,
- imposing selection criteria,
- applying kinematic constraints to reconstructed particle decays,
- saving the resulting information to a flat ROOT TTree for further analysis (analysis output).

The Analysis module is going to be permanently improved and extended.

2.8 Service tools

Test tools

The geometry validation package includes a tool allowing volume overlap checking (both at the DD4Hep level and in the Geant4 simulation). There is also geometry hierarchy printing and material scanning tools based on DD4Hep.

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

Visualization

There are two visualization programs (Figure 3) relying on DD4Hep component DDEve. The geometry display is just for drawing the detector subsystems in various combinations. The event display also reads and shows events. While the geometry display is rather robust already, the event display is still at the very basic stage, partially due to yet not finalized EDM.

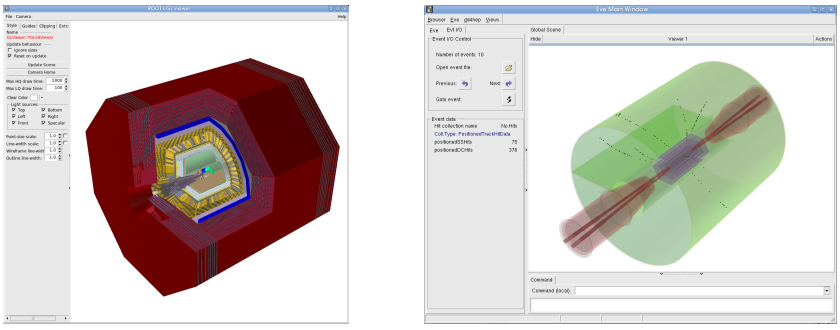

Figure 3. Geometry (left) and Event (right) display tools. A $\psi(4040)$ hadronic decay in central part of the detector is presented at the event display.

Build and version control systems

Build system of the software stack is based on a similar one from the ATLAS[22] experiment. External standard HEP packages are built using LCGCMake[23] infrastructure. The software development workflow utilizes GitLab[24] as a version control and project management system. A continuous integration mechanism, utilizing test tools mentioned above, allowing regular test builds and merge requests processing is going to be introduced.

3 Conclusion

The SCT software development goes on promptly, for both the core framework and subsystem-specific parts, with more experimental groups joining the activity. Some of the framework components are absent yet (for instance, the conditions database), being irrelevant at the present stage of the project.

The Aurora framework now contains all tools minimally required at the present stage of the SCT detector project development:

- set of primary event generators,
- usable in analysis parameterized simulation,
- detector geometry (with at least basic description for all detector elements, and several options for some subsystems),
- full Geant4-based simulation,
- sample digitization module,
- reconstruction modules (from basic to really advanced, depending on subsystem),
- analysis and job configuration tools,
- test and service tools.

This allows us to officially announce the Aurora 1.0.0 release. Actual physics analysis is already possible with the release.

The nearest goals for the software development are:

- implementation of digitization modules for all subsystems,
- further reconstruction improvements, including adoption of some high-level tools, i. e. track finding,
- improvement of detector and event visualization tools. The underlying DDEve has been not actively developed recently, so this is an area of possible backward contribution to DD4Hep,
- distribution of the software via CvmFS[25].

We are looking forward to prepare next major release of the framework till the end of year 2021.

Acknowledgments

We are grateful to the Belle II collaboration for access to the Belle II detector software[26]. We also thank the FCCSW project[27].

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

References

[1] V. V. Anashin et al. (SCT Colaboration), *Super Charm–Tau Factory, Conceptual design report (draft), part one (physics program, detector)*, https://ctd.inp.nsk.su/wiki/images/4/47/CDR2_ScTau_en_vol1.pdf (2018), accessed 2021-02-25

[2] *Key4HEP: Turnkey Software for Future Colliders*, https://github.com/key4hep, accessed 2021-02-26

[3] *The Gaudi project at CERN*, http://gaudi.web.cern.ch/gaudi/, accessed 2021-02-25

[4] *Semantic Versioning*, https://semver.org/, accessed 2021-02-25

[5] *ROOT data analysis framework*, https://root.cern/, accessed 2021-02-25

[6] *YAML: YAML Ain't Markup Language*, https://yaml.org, accessed 2021-02-25

[7] F. Gaede, B. Hegner, P. Mato, J. Phys. Conf. Ser. **898** 072039 (2017), https://github.com/AIDASoft/podio

[8] *EDM4hep*, https://edm4hep.web.cern.ch/, accessed 2021-02-26

[9] *EvtGen Monte Carlo event generator*, https://evtgen.hepforge.org/, accessed 2021-02-25

[10] *Tauola*, http://tauolapp.web.cern.ch/tauolapp/, accessed 2021-02-25

[11] *Photos*, http://photospp.web.cern.ch/photospp/, accessed 2021-02-25

[12] T. Sjöstrand, S. Mrenna, P. Skands, Comput. Phys. Commun. **178** 852 (2008), arXiv:1410.3012, http://home.thep.lu.se/~torbjorn/Pythia.html

[13] A. B. Arbuzov, G. V. Fedotovich, F. V. Ignatov, E. A. Kuraev, A. L. Sibidanov, Eur. Phys. J. C **46**, 689 (2006), arXiv:hep-ph/0504233

[14] G. Balossini, C. M. Carloni Calame, G. Montagna, O. Nicrosini, F. Piccinini, Nucl. Phys. **B758** 227-253 (2006), arXiv:hep-ph/0607181

[15] R. Kleissa, H. Burkhardt, Comput. Phys. Commun. **81** 372 (1994), arXiv:hep-ph/9401333

[16] S. Jadach et al., Comput. Phys. Commun. **130** 260 (2000), arXiv:hep-ph/9912214

[17] A. Buckley et al., Comput. Phys. Commun. 260 107310 (2021), arXiv:1912.08005, http://hepsoftware.org/e/hepmc3

[18] *Detector Description Toolkit for High Energy Physics*, https://dd4hep.web.cern.ch/dd4hep/, accessed 2021-02-25

[19] *Geant4 simulation toolkit*, https://geant4.web.cern.ch/, accessed 2021-02-25

[20] *GenFit Package*, https://github.com/GenFit/GenFit, accessed 2021-02-25

[21] *Acts Common Tracking Software*, https://github.com/acts-project/acts, accessed 2021-02-25

[22] *The ATLAS Experiment*, https://atlas.cern/, accessed 2021-02-25

[23] *LCGCMake*, https://gitlab.cern.ch/sft/lcgcmake, accessed 2021-02-25

[24] *GitLab: The complete DevOps platform*, https://about.gitlab.com/, accessed 2021-02-25

[25] *CernVM File System*, https://cernvm.cern.ch/fs/, accessed 2021-02-25

[26] T. Kuhr, C. Pulvermacher, M. Ritter, T. Hauth, N. Braun, *Computing and Software for Big Science* **3**, Article number: 1 (2018),

[27] *Software for the Future Circular Collider*, http://hep-fcc.github.io/FCCSW/, accessed 2021-02-26