Offline Software Framework
for the Super Tau Charm Facility (STCF)

Wenhao Huang
Shandong University

On behalf of the STCF Software Group

ACAT2021, Dec. 1st, 2021
Outline

- Introduction to STCF
- Overview of STCF Offline Software
- SNiPER framework
- Event Data Model
- Detector Description
- Detector simulation
- Validation System
- Summary
Introduction to STCF

◆ Tau-Charm factory in China has 30 years history
  ⇒ From BEPCI/BESI-II to BEPCII/BESIII
  ⇒ Successful operation, Fruitful physics results
  ⇒ Pursuing higher and higher center-of-mass energy (CME) and Luminosity

◆ STCF is proposed for next Tau-Charm factory in China
  ⇒ CME: 2 – 7 GeV
  ⇒ Luminosity: > 0.5 × 10^{35} cm^{-2} s^{-1} (100 times of BESIII)
  ⇒ BESIII-like detector based on new technologies
  ⇒ Potential to further improve the luminosity and realize polarized beam
  ⇒ More and higher challenge for offline software system
The Offline Software of Super Tau-Charm Facility (OSCAR) was designed and developed for detector design and optimization as well as M.C data production and physics analysis in future.

- Started development based on SNiPER framework from 2018
- Partially based on Key4hep
  - Adopted DD4hep for detector description in 2019
  - Recently move to PODIO for Event Data Model
- Migrated some mature applications from BESIII
SNiPER Framework

- Developed and optimized for non-collider experiments but also suitable for collider experiments
- It is light-weighted and successfully adopted by JUNO (neutrino), LHAASO (cosmic ray), nEXO (neutrinoless double beta decay) …
- The basic unit of applications is Task
  - Application manager
  - Event loop automatically
  - Usually consist of **algorithms, services and sub-tasks**
    - Algorithm is the unit of event loop, usually defined by users.
    - Service provides common functions, like data I/O…
    - Sub-tasks have their own algorithms, services and sub-tasks…
Parallelism in SNiPER

◆ SNiPER Muster (Multiple SNiPER Task Scheduler)
  ➤ Based on Intel Threading Building Blocks (TBB).
  ➤ Create multiple instances of Task.
  ➤ Create TBB-based workers to execute the Task.
    • Each worker would be binded to one thread.
    • One SNiPER Task could be the one on slide 5

Event Data Model based on ROOT

Event data model (EDM) is the core of every HEP experiment’s software framework.

- Defines interface and communication channels between the different framework components.
- Defines the description of physics event.

EDM in OSCAR was defined based on ROOT since 2018.

- EDM for each process and each sub-detector
- Two levels: Event Header and Event
- SmartRef reference class
  - Relationship, associations between files, lazy-loading
- XOD generates C++ codes from xml files.

Optimized EDM based on ROOT

MCEvent

RecEvent
EDM4hep and Podio

- EDM4hep project
  - A common EDM designed for lepton and hadron colliders.
  - Handle unique and non-unique relations between objects of arbitrary data types.
  - Uses plain-old-data (POD) types wherever possible.
    - Simple memory model, performant I/O, vectorization…
  - Memory management
  - Multithread supported
  - Automatic code generation.
  - Adopted by CEPC, FCC.

https://github.com/AIDASoft/podio

Poster about Key4hep: https://indico.cern.ch/event/855454/contributions/4604989/
Event Data Model based on Podio

- Extended the EDM4hep based on PODIO on demand of STCF
- Defined event objects for simulation and reconstruction.
- Build relationships between recon. objects and simulated Objects.
- Automatically generated c++ codes from yaml files
Integration of Podio into OSCAR

Podio memory management could be simplified as

- EventStore to handle event data in memory.
- ROOTReader to input data from disk to memory.
- ROOTWriter to output data from memory to disk.

Similar components in memory management of OSCAR

- DataStore
- Input System
- Output System

Integrated into corresponding components in OSCAR.
Integration of podio into SNiPER

- Input system and Output system are invoked by event loop
  - Finish reading at the begin of event loop
  - Finish writing at the end of event loop

- Provided user interfaces
  - getRWColl( … ) for Writing only
  - getROColl( … ) for Reading only

```cpp
ITDHitCollection* itdhits = getROColl(ITDHitCollection, "ITDHitColl");
ECALHitCollection* ecalhits = getRWColl(ECALHitCollection, "ECALHitColl");
ECALPointCollection* ecalpoints = getRWColl(ECALPointCollection, "ECALPointColl");
```
Geometry Management System (GMS)

- Detector description based on DD4hep
- Each sub-detector has its geometry description
- Flexible combinations of different scenario and support single/full detector simulation

Structure of the geometry parameters repository

Poster about GMS: https://indico.cern.ch/event/855454/contributions/4596406/
Geometry description of sub-detector

DTOF  ECAL  MDI
MUC  RICH  VTD
Detector Simulation

- Design of detector simulation system
  - Lightweight, be easily migrated from standalone application

- The component for sub-detector simulation can be easily integrated for full detector simulation
  - Detector description, user actions…
Adoption of DD4hep

- DD4hep has been integrated into OSCAR
- Single source of information support detector description, simulation and reconstruction with a same service
  - DDG4 for delivering detector geometry to Geant4
  - DDRec for delivering detector geometry to reconstruction algorithms
  - DDXMLSvc: the unified interface to DD4hep, including DDG4 and DDRec
- OSCAR provides a macro to generate sensitive detector according to string: `DECLARE_SENSDET(DetName)`
DDXMLSvc for Sim. and Rec.

- Most geometry information could be provided

```cpp
SniperPtr<DDXMLSvc> ddsvc(this->getParent(), "DDXMLSvc");  //Get the service
m_PhysicalVolume = ddsvc->getPhyVol();                       //Get physical volume
```

- Developed the GeometrySvc of sub-detector for reconstruction

```cpp
SniperPtr<SubGeoSvc> subsvc(this->getParent(), "SubGeoSvc");

m_subdata = subsvc->getSubData("Detname");
```

- DDXMLSvc provide basic info
  - Physical volume, magnetic field...

- SubGeoSvc provide info for specific detector...
Sub-detector and full simulation

- Now every sub-detector has its geometry description
  - Run simulation of one sub-detector

```python
factory = task.createSvc("FullSimFactory/FullFactory")
factory.property("AnaMgrList").set(["GeneratorMgr", "ECALAnaMgr"])  # Run simulation of full detector

factory = task.createSvc("FullSimFactory/FullFactory")
factory.property("AnaMgrList").set(["GeneratorMgr", "ECALAnaMgr", "ITDAnaMgr", "MDCAnaMgr", "FTOFAnaMgr", "RICHAnaMgr", "MUCAnaMgr",])
```
Automated Validation System

◆ An automated validation system is being developed for software validation at different levels
  ➞ Unit test, integrated test, performance test, physical validation etc.

◆ A powerful toolkit is developed for building software validation workflow
  ➞ Provide interfaces to define and run unit tests
  ➞ Support various detectable failures (log errors, memory leaking, ...)
  ➞ Support performance profiling
  ➞ Support results validation based on statistical methods
Summary

◆ OSCAR is developed for STCF based on SNiPER framework
◆ Several state-of-art software and tools in the HEP community are adopted, such as DD4hep, Podio, camke, gitlab
◆ The detector simulation Chain has been set up for full detector simulation
◆ Automatically Validation System are under development
Thanks for your attention!

Any questions are welcome!