Towards a Turnkey Software Stack for HEP Experiments

André Sailer, Gerardo Ganis, Pere Mato, Marko Petrič, Graeme Stewart
CERN

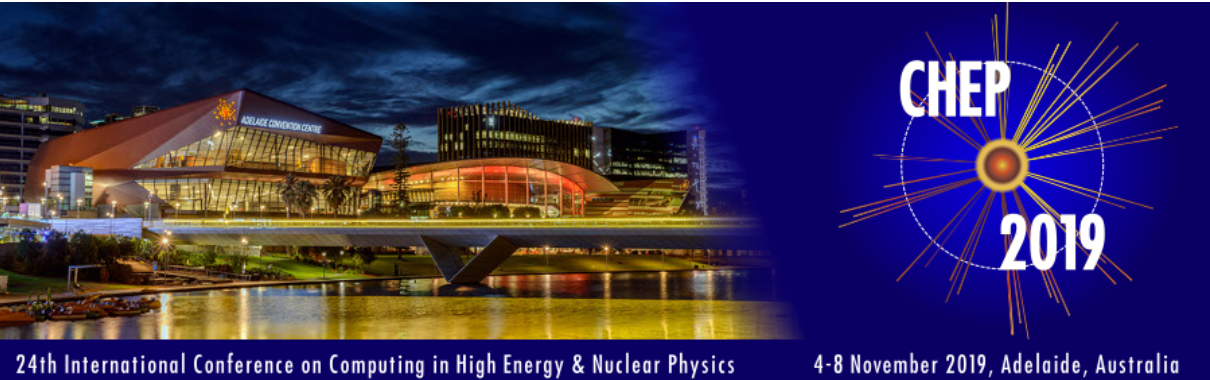
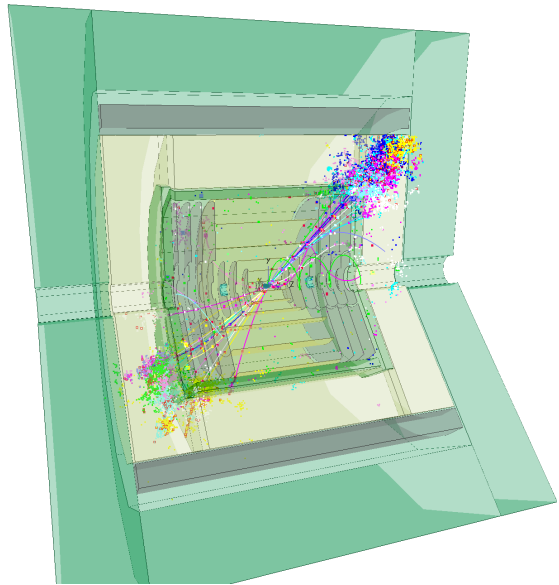


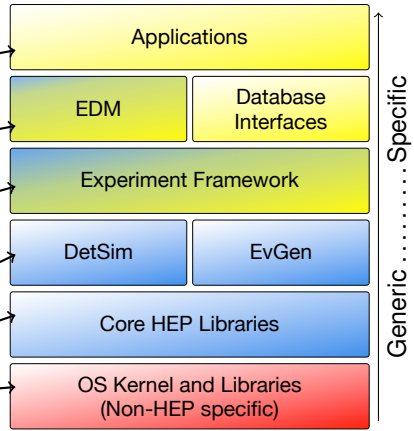
Table of Contents



A typical HEP Software Stack

Applications usually rely on large number of libraries, where some depend on others

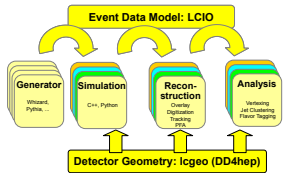
- Interfaces to tracking and reconstruction libraries (PandoraPFA, ACTS)
- (More or less) experiment specific event data model libraries
- Experiment core orchestration layer, which controls everything else: Marlin, Gaudi, CMSSW, AliRoot
- Packages used by many experiments: DD4hep, Pythia, ...
- Usual core libraries (ROOT, Geant4, CLHEP, ...)
- Non-HEP libraries: boost, python, cmake ...



Towards a Turnkey Software Stack

- Members of the FCC, ILC, CEPC, SCT, CLIC, LHC communities met for a Future-Collider-Software Workshop in Bologna on June 12&13 https://agenda.infn.it/event/19047/
  - Reached an Agreement to create common event data model and common turnkey software stack

- The Vision: The turnkey stack connects and extends the individual packages towards a complete data processing framework
  - Major ingredients: Event Data Model, Geometry Information, Processing Framework
  - Sharing common components reduces overhead for all users
  - Should be easy to use for librarians, developers, users
    - ⋆ easy to deploy, extend, set up
  - full of functionality: plenty of examples for simulation and reconstruction of detectors



iLCSoft components, but general scheme of ingredients applies

Turnkey Software Stack part of

- EP R&D Programme; chapter 10.7
- AIDA++ submission: software work package; partners: CERN, DESY, INFN

Interoperability of Packages

The right interoperability point between packages varies, but choosing it correctly provides great *quality of life* for developers and users

- Level 0 – Common Data Formats
  - E.g.: HepMC event records, LCIO, GDML, ALFA Messages
- Level 1 – Callable Interfaces
  - Details are important: error/exception handling, thread safety, library dependencies, runtime setup
- Level 2 – Introspection Capabilities
  - E.g.: PyROOT to interact with any ROOT (C++) class via the python interpreter
- Level 3 – Component Model
  - Software components of a *common framework* offer maximum re-use
  - *standard* way to configure components, logging, object lifetime and ownership, plug-in mechanism
  - Requires adoption of *single* framework

More on this in the presentation by G. Stewart at Bologna WS on Common Software Tools

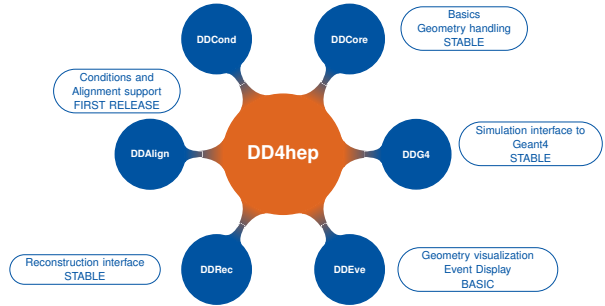
Data Processing Framework

- Data processing frameworks are the skeleton on which HEP applications are built
  - Some parts of the glue between execution of algorithms and frameworks is rather specific (usually the *services*; logging, histogram store, data store), so there is a huge advantage of using one framework
- `Marlin` very successfully used in LC community to run on different detector models or test beam data
  - Very nice features for automatic steering file configuration, ease of use
  - Concurrency support under development (See MarlinMT - parallelising the Marlin framework)
- `Gaudi` is used by LHCb, ATLAS, SCT, FCC (A software framework for FCC studies: status and plans)
  - Supports concurrency, access to heterogeneous resources, larger developer community

Based on considerations such as completeness, portability to various computing resources, architectures and accelerators, support for task-oriented concurrency and current adoption, it was agreed the best candidate to date is the Gaudi framework.

Geometry Information: DD4HEP

- Complete Detector Description
  - Providing geometry, materials, visualization, readout, alignment, calibration...
- Single source of information → consistent description
  - Use in simulation, reconstruction, analysis
- Supports full experiment life cycle
  - Detector concept development, detector optimization, construction, operation
  - Facile transition from one stage to the next

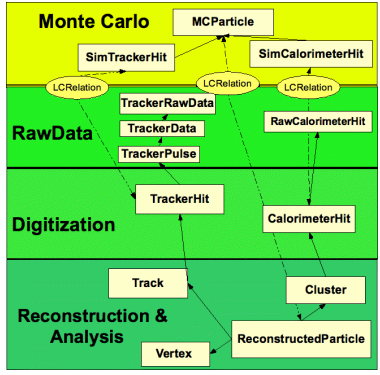


- DD4HEP already in use by CLIC, FCC and many more, easy choice for geometry provider
- Latest developments: DD4HEP: a community driven detector description tool for HEP

Event Data Model: PODIO and EDM4hep

For a high degree of interoperability, EDM4HEP will provide a common event data model

- Using PODIO to manage the EDM (described by yaml) and easily change the persistency layer (ROOT, HDFS, ...)
- Develop an *alpha* version of EDM4HEP based on LCIO and FCC-edm in the next few months
- See how best to move forward
- Input from other interested parties highly welcome
- EDM4HEP working group:
  - https://indico.cern.ch/category/11461/
  - http://github.com/HSF/edm4hep
- See also:
  PODIO: recent developments in the Plain Old Data EDM toolkit



LCIO EDM Objects

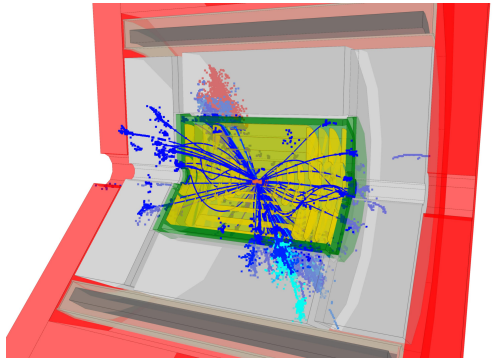
Packaging: Spack

- For ease of use for librarians and developers, need to be able to build any and all pieces of the stack with minimum effort
- Go beyond sharing of build results to sharing of build recipes
- HSF packaging working group pointing towards adoption of Spack
- See Modern Software Stack Building for HEP

Evolution of the CLIC Reconstruction

CLIC Reco Evolution: Adiabatic Changes

- Full CLIC reconstruction implemented in iLCSoft
- While transitioning to KEY4HEP, need to be able to keep running the CLIC reconstruction
- Switch components one by one, validate changes
  - Geometry provided by DD4HEP, no changes needed
  - Move framework from Marlin to Gaudi: wrap existing processors
  - Move from LCIO to EDM4HEP
  - Replace wrapped processors with native Gaudi algorithms
- Incidentally will make iLCSoft functionality available to other users of the stack



Marlin & Gaudi

Apart from some naming conventions, very similar ideas in the two frameworks*

	Marlin	Gaudi
language	`c++`	`c++`
working unit	Processor	Algorithm
configuration language	XML	Python
set up function	`init`	`initialize`
working function	`processEvent`	`execute`
wrap up function	`end`	`finalize`
Transient data format	`LCIO`	anything

- To start using Gaudi: use a generic wrapper around the processors.
- Prototype: https://github.com/andresailer/GMP
- Read LCIO files and pass the `LCIO::Event` to our processors

*Of course subtle differences emerge

Wrapper Configuration

- Translate the `XML` to `python`, using a stand alone python script
- Pass arbitrary number, types, and names of parameters to the processor

Marlin/XML

```
<processor name="VXDBarrelDigitiser" type="DDPlanarDigiProcessor">
  <parameter name="SubDetectorName" type="string">Vertex </parameter>
  <parameter name="IsStrip" type="bool">false </parameter>
  <parameter name="ResolutionU" type="float"> 0.003 0.003 0.003 0.003 0.003 0.003 </parameter>
  <parameter name="ResolutionV" type="float"> 0.003 0.003 0.003 0.003 0.003 0.003  </parameter>
  <parameter name="SimTrackHitCollectionName" type="string" lcioInType="SimTrackerHit">VertexBarrelColl
  <parameter name="SimTrkHitRelCollection" type="string" lcioOutType="LCRelation">VXDTrackerHitRelation
  <parameter name="TrackerHitCollectionName" type="string" lcioOutType="TrackerHitPlane">VXDTrackerHits
  <parameter name="Verbosity" type="string">WARNING </parameter>
</processor>
```

Wrapper Configuration

- Translate the XML to python, using a stand alone python script
- Pass arbitrary number, types, and names of parameters to the processor

Gaudi/Python

```
VXDBarrelDigitiser = MarlinProcessorWrapper("VXDBarrelDigitiser")
VXDBarrelDigitiser.OutputLevel = WARNING
VXDBarrelDigitiser.ProcessorType = "DDPlanarDigiProcessor"
VXDBarrelDigitiser.Parameters = [
                                "IsStrip", "false", END_TAG,
                                "ResolutionU", "0.003", "0.003", "0.003", "0.003", "0.003", "0.003", END
                                "ResolutionV", "0.003", "0.003", "0.003", "0.003", "0.003", "0.003", END
                                "SimTrackHitCollectionName", "VertexBarrelCollection", END_TAG,
                                "SimTrkHitRelCollection", "VXDTrackerHitRelations", END_TAG,
                                "SubDetectorName", "Vertex", END_TAG,
                                "TrackerHitCollectionName", "VXDTrackerHits", END_TAG
                                ]
```

Configuration: Control flow

- XML `execute` section translated to a python list

```xml
<execute>
  <processor name="MyAIDAProcessor"/>
  <processor name="EventNumber" />
  <processor name="InitDD4hep"/>
  <processor name="Config" />
  <!-- ... -->
</execute>
```

```python
algList = []
algList.append(lcioReader)
algList.append(MyAIDAProcessor)
algList.append(EventNumber)
algList.append(InitDD4hep)
algList.append(OverlayFalse)
algList.append(VXDBarrelDigitiser)
#...
```

Changes in iLCSoft

Surprisingly little changes needed in `Marlin`

- Make `marlin::Processor::setParameters` and `marlin::Processor::setName` `public`
- Actually make the `Marlin` `EventSelector` part of the namespace to avoid clash with `EventSelector` from `Gaudi`
- Make it possible to call the `marlin::ProcessorEventSeeder` from the wrapper; move to functions from `private` to `public`

Summary

- Studies for future experiments can benefit from having a complete software stack available to them
- Starting from CLIC and FCC studies, develop a turnkey software stack: KEY4HEP
- Using established solutions: ROOT, Geant4, DD4HEP, Gaudi, ...
- Investigate, develop, or adopt where beneficial or necessary: EDM4HEP, Spack
- On the way to run CLIC reconstruction with Gaudi; EDM4HEP close to alpha release
- Interested parties are welcome to participate