Integrating iLCSoft into Key4hep

LCWS2021

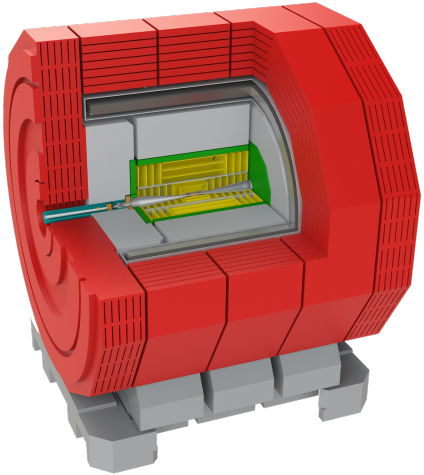
Plácido Fernández Declara

March 15, 2021

CERN

iLCSoft for Key4hep

- Bring existing, battle-tested algorithms and software from iLCSoft to future colliders.
- Integrate in a smooth non-disruptive way.
- Run algorithm chains that can contain previous, current and future algorithms.
- Added support for interfaces and converters that allow for the integration.



k4MarlinWrapper[2]

- Part of the Key4hep[1] project:
  - Brings analysis and reconstruction to the common software stack.
- *Marlin* Processors functionality made available in Key4hep through the *Gaudi* framework.
- It contains the necessary interfaces to deal with *Marlin* formats to be run from *Gaudi* algorithms.
  - Wrapper around Marlin Processors.
  - XML steering file to Python options file converter.
  - In-memory converters between event data models.
- Marlin source code is kept intact, and can be called on demand.

[1] `https://github.com/key4hep/`
[2] `https://github.com/key4hep/k4MarlinWrapper`

Dependencies

k4MarlinWrapper can be built against the Key4hep CVMFS view. Main dependencies:

- **Gaudi**: to wrap Marlin processors and run the algorithms.
- **Marlin**: to run the underlying processors.
  - It will eventually disappear when only Gaudi Algorithms are used.
- **LCIO**: Event Data Model input/output used by Marlin.
- **EDM4hep**: Event Data Model input/output to be used across the framework.
  - Other event data models could be integrated.
- **k4FWCore**, **k4LCIOReader**, **podio**: leveraging synergies between other Key4hep packages and related.

Other general dependencies:

- **ROOT**, **Boost**

Configuration and running

- Config and running done via Python file as with the Gaudi Framework.
- Processor parameters defined for each instance, and list algorithms configured.
- On algorithm initialization of Marlin Processors, the `MARLIN_DLL` environment variable is used to load the necessary libraries.

```
MyTPCDigiProcessor = MarlinProcessorWrapper("MyTPCDigiProcessor")
MyTPCDigiProcessor.OutputLevel = INFO
MyTPCDigiProcessor.ProcessorType = "DDTPCDigiProcessor"
MyTPCDigiProcessor.Parameters = [
            "DiffusionCoeffRPhi", "0.025", END_TAG,
            "DiffusionCoeffZ", "0.08", END_TAG,
            "DoubleHitResolutionRPhi", "2", END_TAG,
            "DoubleHitResolutionZ", "5", END_TAG,
            "HitSortingBinningRPhi", "2", END_TAG,
            "HitSortingBinningZ", "5", END_TAG,
            "MaxClusterSizeForMerge", "3", END_TAG,
            "N_eff", "22", END_TAG,
            # ...
            ]
algList.append(MyTPCDigiProcessor)
```

Listing algorithms

Incomplete options file example:

```
inp = PodioInput('InputReader')
InitDD4hep.ProcessorType = "InitializeDD4hep"
VertexFinder.ProcessorType = "LcfiplusProcessor"
JetClusteringAndRefiner.ProcessorType = "LcfiplusProcessor"
MakeNtuple.ProcessorType = "LcfiplusProcessor"
MyLCIOOutputProcessor.ProcessorType = "LCIOOutputProcessor"
output = PodioOutput("PodioOutput", filename = "my_output.root")

algList.append(input)
algList.append(InitDD4hep)
algList.append(VertexFinder)
algList.append(JetClusteringAndRefiner)
algList.append(MakeNtuple)
algList.append(MyLCIOOutputProcessor)
algList.append(output)
```

XML to Python converter

- A converter from XML steering file to Python options file is available as a Python script.
- It produces the list of Gaudi algorithms, including optional Processors.
  - These are left as commented algorithms that need to be manually uncommented by the user.
  - A comment is also included to indicate its configuration.
  - `# algList.append(MyFastJetProcessor) # Config.OverlayNotFalse`
- It now includes *Constants* parsing from the XML
  - It lists the `CONSTANTS =` to be modified by the user
  - These are replaced in the processors with String substitution: `"%(DD4hepXMLFile_subPath)s" % CONSTANTS`
  - It now supports lists of arguments in the constants as well
- `Marlin -x` can create a steering file containing all the parameters for the known processors. This can be converted to python.

Managing input and output

- Input and Output can be managed by using *LcioEvent* Algorithm for input and the *LCIOOutputProcessor* wrapped by *k4MarlinWrapper*
  - Allow to run as with Marlin
- *Podio* can be used to read and write collections.
  - DataHandle used under the hood to take care of the collections.
  - *PodioInput* and *PodioOutput* can be used for this purpose.

```
read = LcioEvent()
read.Files = ["../test/inputFiles/muons.slcio"]

Output_DST = MarlinProcessorWrapper("Output_DST")
Output_DST.ProcessorType = "LCIOOutputProcessor"
Output_DST.Parameters = [...]
```

```
from Configurables import PodioInput, PodioOutput
inp = PodioInput('InputReader')
inp.collections = [
    'ReconstructedParticles',
    'EFlowTrack']

out = PodioOutput("PodioOutput", filename = "my_output.
out.outputCommands = ["keep *"]
```

EDM4hep to LCIO conversion

- In memory conversion between EDM4hep and LCIO needed to run Marlin Processors and Edm4hep based Gaudi Algorithms at the same time
- Uses *k4FWCore*[3] to read the input collections indicated in the options file.

```
from Configurables import k4DataSvc, ToolSvc, MarlinProcessorWrapper, EDM4hep2LcioTool
theFile= 'edminput.root'
evtsvc = k4DataSvc('EventDataSvc')
evtsvc.input = theFile

from Configurables import PodioInput
inp = PodioInput('InputReader')
inp.collections = [
    'ParticleIDs', 'ReconstructedParticles', 'EFlowTrack'
]
algList.append(inp)
```

[3] https://github.com/key4hep/k4FWCore

EDM4hep to LCIO conversion

- Converter implemented in *k4MarlinWrapper* as a Gaudi Tool
- Configured in the options file indicating which processor needs to use the Tool to convert the EDM4hep event to LCIO format
- Events are read through the DataHandle from k4FWCore; these are converted and registered in the Transient Event Store (TES) to make it available to the framework.

```
MyFastJetProcessor = MarlinProcessorWrapper("MyFastJetProcessor")
# ...
edmConvTool = EDM4hep2LcioTool("EDM4hep2lcio")
edmConvTool.EDM2LCIOConversion = [
    "Track", "EFlowTrack", "EFlowTrackConv",
    "ReconstructedParticle", "ReconstructedParticles", "TightSelectedPandoraPFOs"]
# ...
MyFastJetProcessor.EDMConversionTool=edmConvTool
# ...
algList.append(MyFastJetProcessor)
```

LCIO to EDM4hep conversion

- Converter from LCIO format to EDM4hep needed:
  - Interoperability between different algorithms.
  - Write output in EDM4hep format.
- Actual conversion integrated from k4LCIOReader[4], part of Key4hep.
  - k4LCIOReader meant to be used to read input files.
  - Adapted to read in-memory collections and convert them.
- Integrated with k4FWCore to seamlessly integrate the converted types: Podio output used to write the collections back.
- Implemented as a Gaudi Tool: can be attached to any Gaudi algorithm from k4MarlinWrapper.

[4] `https://github.com/key4hep/k4LCIOReader`

CLIC reconstruction

It successfully computes the full CLIC reconstruction

- CLIC reconstruction computes a sequence with different overlays, digitisers, reconstruction, PFO selectors, trackers, vertex finding algorithms, and others.
  - Complete sequence instantiated from k4MarlinWrapper delivering same results.
  - Constants used in converted version.
- Input is converted with the updated XML to Python converter.
- Configurable optional processors by the users.

```
from Gaudi.Configuration import *
CONSTANTS = {'BCReco': "3TeV",}
parseConstants(CONSTANTS)
# ...
read = LcioEvent()
InitDD4hep = MarlinProcessorWrapper("InitDD4hep")
Config = MarlinProcessorWrapper("Config")
VXDBarrelDigitiser = MarlinProcessorWrapper("VXDBarrelD
VXDEndcapDigitiser = MarlinProcessorWrapper("VXDEndcapD
# ...
algList.append(InitDD4hep)
algList.append(Config)
# algList.append(OverlayFalse)
# algList.append(Overlay350GeV_CDR)
algList.append(VXDBarrelDigitiser)
algList.append(VXDEndcapDigitiser)
# ...
```

LCFI+ algorithm

- Group of algorithms to perform vertex finding, jet finding and flavour tagging for linear colliders.
- Different Event Data Models needed to use the group of algorithms in different experiments (i.e. FCC-ee)
  - LCFI+ implemented in Marlin using LCIO.
  - Generated input in EDM4hep, e.g., by Delphes
  - Output expected in EDM4hep.
- Successfully implemented with the in-memory converters.

```
inp = PodioInput('InputReader')
inp.collections = ['ReconstructedParticles', 'EFlowTrack']
# ...
edmConvTool = EDM4hep2LcioTool("EDM4hep2lcio")
edmConvTool.EDM2LCIOConversion = [...]
# ...
InitDD4hep.ProcessorType = "InitializeDD4hep"
InitDD4hep.EDMConversionTool=edmConvTool
# ...
lcioConvTool = k4LCIOReaderWrapper("LCIO2EDM4hep")
lcioConvTool.LCIO2EMD4hepConversion = [...]
# ..
JetClusteringAndRefiner.ProcessorType = "LcfiplusProcessor"
JetClusteringAndRefiner.LCIOConversionTool=lcioConvTool
# ...
out = PodioOutput("PodioOutput", filename = "my_output.root")
out.outputCommands = ["keep *"]
```

SCTau integration

- Integration with k4MarlinWrapper would provide all Marlin algorithms to be used in the SCTau (Super-Charm-Tau) software framework (Aurora).
  - Aurora is Gaudi based, so k4MarlinWrapper shares most of the components.
  - Different versions of core packages impede an easy integration.
  - k4MarlinWrapper can be included as an external.
- Geometry and sensitive detector TPC needs to be adapted from iLC to SCT.
- Event Data Model differs from EDM4hep (or LCIO)
  - To be rebased from EDM4hep to be compatible.

Conclusions and future directions

- *k4MarlinWrapper* has been improved and extended to support more use cases.
  - Added converters and interfaces, extended functionality, bugs fixed.
  - Converters put into test by these use cases.
- It has been successfully used for complete CLIC reconstruction and LCFI+ algorithm.
  - Integration with SCTau is ongoing: EDM and framework integration ongoing.[5]
- EDM converters: more types will be supported in the future

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