CLICSoft transition to EDM4hep

EP R&D Software Working Group Meeting

Plácido Fernández Declara, Valentin Volkl, André Sailer

September 23, 2020

CERN

GMP Wrapper

- The Gaudi-Marlin-Processors Wrapper project brings *Marlin* functionality to *Gaudi* framework, smoothly.
- It creates interfaces *(wraps)* around Marlin Processors, encapsulating them in Gaudi Algorithms.
- Current Marlin source code is kept intact, and it is just called on demand from the Gaudi Framework.

	Marlin	Gaudi
Language	C++	C++
Working unit	Processor	Algorithm
Config. language	XML	Python
Set-up function	init	initialize
Working function	process	execute
Wrap-up function	end	finalize
Transient Data Format	LCIO	anything

GMP Wrapper now

- Bugs were fixed, a manual (`README.md`) was included with instructions to compile, configure, run and test.
- Updated and modernization of the code base.
- Running examples are included as tests.
- A recipe to build it with Spack is also part of the *k4-spack* repo.
- It was included as part of Key4hep, moving there the repo[1].
- CI is now included with GitHub Actions, checking syntax (`clang-format`), and running two basic functionality tests.

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

Dependencies

GMP Wrapper can be built against an iLCSoft installation + Gaudi. 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
  - Can be changed, for EDM4hep i.e.

Other dependencies:

- ROOT, Boost

Or simply[2]:

- `spack install key4hep-stack`

[2] https://key4hep.github.io/key4hep-doc/spack-build-instructions/README.html

GMP Wrapper configuration and running

Configuring and running the wrapper is done as in Gaudi, through a Python file:

- An algorithm list is filled with wrapped Marlin Processors.
- Processors parameters are defined for each instance, defining the Marlin processor to load and list of parameters and values
    - Converter for Marlin XML configuration files exists

On algorithm initialization of a Marlin Processor, `MARLIN_DLL` environment variable is used to load the necessary libraries.

GMP configuration example

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

Testing

Added testing with `ctest`:

- Simple test that runs some Marlin Processors: AidaProcessor -> InitDD4hep -> VXDBarrelDigitiser
- `muon.slcio` is used for input, without hits.
- Second test generates an input file with `ddsim`
- It runs a similar list of algorithms with actual hits
- Output checks for regex with `INFO Application Manager Terminated successfully`

```
    ddsim \
        --steeringFile $ILCSOFT/ClicPerformance/HEAD/clicConfig/clic_steer.py \
        --inputFiles $ILCSOFT/ClicPerformance/HEAD/Tests/yyxyev_000.stdhep -N 4 \
        --compactFile $ILCSOFT/lcgeo/HEAD/CLIC/compact/CLIC_o3_v14/CLIC_o3_v14.xml \
        --outputFile $GMP_tests_DIR/inputFiles/testSimulation.slcio
```

CLIC reconstruction

It successfully computes the full CLIC reconstruction:

- The CLIC reconstruction computes a sequence that includes different Overlays, Digitisers, reconstruction, tracker and vertex finding algorithms.
- Using the updated converter, `clicReconstruction.xml` can be translated to `clicReconstruction.py`.
- The converter add all algorithms to the list, and leaves the configurable ones commented.

Future directions

- Move from LCIO to EDM4HEP.
  - Converter available in K4LCIOReader [3]
- Replace wrapped Marlin Processors by actual Gaudi Algorithms.
  - Benefit from the different functionalities Gaudi offers
  - Use multi-threaded/functional Gaudi, for the future
  - Seamlessly integrate for other users of Key4hep
- Start using it in real scenarios to test how resilient it is.
- How to approach the transition?
  - Gradual conversion from Marlin Processors to Gaudi Algorithms
  - Transition to EDM4hep, before Processors conversion?
  - Conversions during runtime?

[3] https://github.com/ihep-sft-group/K4LCIOReader/blob/master/src/K4LCIOConverter.cc