

Evolution of iLCSoft

Towards the Common Software Stack Key4HEP

André Sailer

CERN-EP-LCD

CLIC Detector and Physics Collaboration Meeting
August 27–28, 2019

Table of Contents



1 Adapting the CLIC Reconstruction to Gaudi

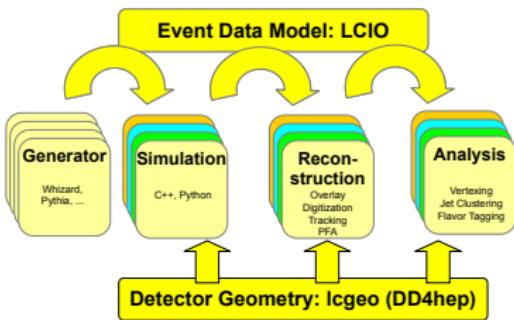
- Marlin & Gaudi
- Reading LCIO Events
- Re-Using Existing Processors
- E(DM)volution
- Implication for User Processors

2 Current Status

3 Conclusion

Adiabatic Changes

- While transitioning to Key4HEP, need to be able to keep running the 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



Apart from some naming conventions, very similar ideas in the two frameworks – glossing over technical details

	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
- Read LCIO files and pass the LCIO::Event to our processors

LCIO Event “Algorithm”



- Reads LCIO files and places the `LCIO::Event` in the `gaudi::DataStore`
- `LCIO::Event` can be used for input and output as is the case in Marlin

Marlin Processor Wrapper



Need one `gaudi::algorithm` to run any `Marlin::Processor`

- **initialize**

- ▶ get processor instance of requested `ProcessorType` from `marlin::ProcessorMgr`
- ▶ prepare `marlin::StringParameters` object
- ▶ Call `Processor::setName` and `Processor::setParameters`

- **execute:**

- ▶ get `LCIO::Event` from `gaudi::DataStore`, call `processor->processEvent`

- **finalize:**

- ▶ call `processor->end`

Prototype: <https://github.com/andresailer/GMP>

Wrapper Configuration



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

```
<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 name="ResolutionV" type="float"> 0.003 0.003 0.003 0.003 0.003 0.003
  <parameter name="SimTrackHitCollectionName" type="string" lcioInType="SimTracker"
  <parameter name="SimTrkHitRelCollection" type="string" lcioOutType="LCRelation">
  <parameter name="TrackerHitCollectionName" type="string" lcioOutType="TrackerHit"
  <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

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

Wrapper Configuration



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

```
<processor name="Refit" type="RefitFinal">
  <parameter name="EnergyLossOn" type="bool"> true </parameter>
  <parameter name="InputRelationCollectionName" type="string" lcioInType="LCRelation">
    SiTracks_Refitted_Relation
  </parameter>
  <parameter name="OutputTrackCollectionName" type="string" lcioOutType="Track">
    SiTracks_Refitted
  </parameter>
  <parameter name="ReferencePoint" type="int"> -1 </parameter>
  <parameter name="SmoothOn" type="bool"> false </parameter>
  <parameter name="extrapolateForward" type="bool"> true </parameter>
  <parameter name="MinClustersOnTrackAfterFit" type="int">3 </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

```
Refit = MarlinProcessorWrapper("Refit")
Refit.OutputLevel = WARNING
Refit.ProcessorType = "RefitFinal"
Refit.Parameters = [
    "EnergyLossOn", "true", END_TAG,
    "InputRelationCollectionName", "SiTrackRelations", END_TAG,
    "InputTrackCollectionName", "SiTracks", END_TAG,
    "Max_Chi2_Incr", "1.79769e+30", END_TAG,
    "MinClustersOnTrackAfterFit", "3", END_TAG,
    "MultipleScatteringOn", "true", END_TAG,
    "OutputRelationCollectionName", "SiTracks_Refitted_Relation", END_TAG,
    "OutputTrackCollectionName", "SiTracks_Refitted", END_TAG,
    "ReferencePoint", "-1", END_TAG,
    "SmoothOn", "false", END_TAG,
    "extrapolateForward", "true", END_TAG
]
```

Configuration: Control flow



- XML execute section translated to a python list
- Can use python for string substitution, static if/else, loops, ...

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

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

Need better understanding of Gaudi for dynamic control flow

Changes in iLCSoft



Very minimal changes needed in Marlin

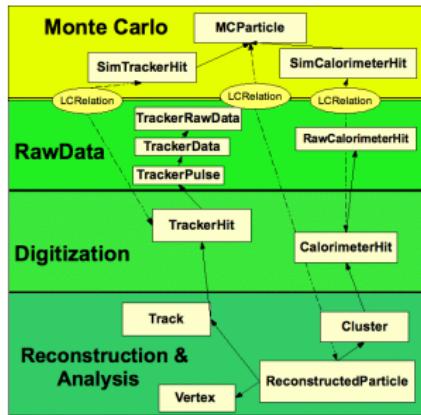
- Make `marlin::Processor::setParameters` and
`marlin::Processor::setName` public
- Rename `marlin::EventSelector` to avoid conflict with
`gaudi::EventSelector`

EventDataModel: EDM4HEP, podio, plcio



Use a common event data model to allow high degree of integration

- Using podio to manage the EDM and easily change the persistency layer
- Create *adapters* to migrate the existing LCIO data to the new EDM



Implication for User Processors



- As long as we write out LCIO files: None
 - ▶ Users can still run Marlin and their own processors on the output files
- Can also run processors inside Gaudi using the wrapper and steering file conversion tool
- When we start transitioning to EDM4HEP
 - ▶ Bi-directional in memory adapters between EDM4HEP and LCIO?
 - ▶ Some work to adapt processors to `gaudi::DataStore` and EDM4HEP

Current Status

- Run track reconstruction for CLIC
 - ▶ Missing dependencies in the environment used to build Gaudi to run the full chain

To do:

- Add possibility to load processors on demand a la MARLIN_DLL
- Deal with errors and exceptions from marlin processors
- Treat Marlin processRunHeader, check functions
- Make developments available beyond Marko's and my PC, with the help of the HSF packaging working group

Key4HEP Benefits for CLIC



- Benefit from framework developments of the LHC experiments by adopting the Gaudi Framework
 - ▶ Multi-threading schedulers
 - ▶ Access to heterogeneous resources
- Focus on modernisation and improvements of our algorithms
- Find new users for the wealth of functionality present in iLCSoft
- Make use of algorithms interfaced with Gaudi