ILC Framework for Belle II

Kolja Prothmann,
Christian Kiesling,
Andreas Moll,
Frank Simon
Max-Planck-Institut für Physik
Zbynek Drasal
Charles University Prague
Outline

- Experiments using ILC software
- Overview of ILC framework
  - Data model: LCIO
  - Geometry and particle transport: Mokka / Gear
  - Digitization and Reconstruction: Marlin / Marlin GUI
  - Event Display: GeV
- Discussion
- Conclusion

available on: /bwf/g67home/kolja/tutorial
Experiments using the ILC Framework

**CALICE**
Calorimeter test experiment
data taking ~100 Mio events
~1 year of H1 data

**Eudet project**
Test experiment
For tracking
Detectors (TPC, Sidet)

**ILD**
LOI Letter of Intent
For their ILD detector at ILC
MonteCarlo production
~50 Mio events

**CLIC: Simulation Studies**
Manpower on the ILC Framework

- ILC Community Europe
  - Frank Gaede
  - Steve Aplin
  - 2 Post Docs
  - France → Mokka
  - Czech Republic (Prague)
  - Austria (Vienna)
  - Japan (ILC community)
  - CERN CLIC

- LOI done now 1 year of framework development until TDR
  - ROOT/IO will be implemented

If framework is chosen for Belle II:
  → Framework team within Belle II needed
Data Model

Persistence layer:
- stores objects in files
- designed to be exchanged easily

Data Model:
Objects and Relations

All languages see the same API (same Objects)

C++  Java  Fortran
LCIO

SIO  ROOT/IO (planned)
Data Model: details

- bidirectional Relations between LCOObjects
  - one to one
  - one to many
  - many to many
- suitable for DAQ and Raw Data
- Model can be modified for Belle II (for example PID)
Schematics of the ILC Framework

Simulation

- Pythia (Generator)
- Mokka (Geant4)
- HEPEvt (ASCII) - Generated events

Reconstruction

- LCIO (persitency data model)
- Marlin
  - MaterialDB
  - Digitization
  - Tracking
  - Optimization
- GEAR (XML) - Geometry information
- ROOT
- LCCD

MySQL
Panther banks to HEPEvt converter

- new BASF module (hepevt)
- Extracts the Monte Carlo information generated by `evtgen`
- reads the `Gen_HEPEVT` table from the panther banks
- writes out a std. `HEPEvt` file which can be used in `Mokka`

→ Bridge from BASF to ILC framework!
Geometry and Particle Transport: Mokka
GEAR – GEometry Api for Reconstruction

- Provides a simplified geometry description for reconstruction
- Provides separation between Data(LCIO) & Geometry
- at the moment data is stored in a separate XML file
  - The datasource can be changed to Belle II needs e.g. use a database
Mokka – Belle II Geometry - 1

• Mokka model: *complete Belle II Tracker* (beam pipe + PXD + SVD + CDC)

  • **Beam pipe**: cylindrical onion-like structure
    - inner golden layer + inner Be wall + cooling gap (paraffin) + outer Be wall

  • **PXD**: 2 layers of Si pixel detectors – DEPFETs
    - active part: layers → ladders → Si sensors (50µm)
    - passive part: Si rims (450µm) + 12 switchers (300µm)

<table>
<thead>
<tr>
<th></th>
<th>R [mm]</th>
<th># ladders</th>
</tr>
</thead>
<tbody>
<tr>
<td>PXD layer 1</td>
<td>18</td>
<td>10</td>
</tr>
<tr>
<td>PXD layer 2</td>
<td>22</td>
<td>12</td>
</tr>
</tbody>
</table>
Mokka – Belle II Geometry - 2

- **SVD**: 4 layers of Si strip detectors (DSSDs) in barrel part
  - organized in stagger-like structure
  - active part: layers → ladders → Si sensors (300µm)

- **CDC**: Al cylinder with cone-shaped inner parts (as Belle)
  - active medium: gas He/C$_2$H$_6$ (50:50)
  - uses Gaussian smearing as digitization
  - geometry as of December 2008
Marlin Processors

- Digitizers (PXD, SVD and CDC)
- Stand alone tracking (PXD+SVD or CDC)
- Full tracking (PXD+SVD+CDC)
- Other Processors available:
  - LCFIVertex (Vertexing)
  - Pandora (Particleflow: ILC specific)
  - Curlkiller, ..... 
- Analysis Processor (Knowledge of Belle would go in there)
How to create a Marlin steering file?

• Use the MarlinGUI

• Let Marlin create a sample XML steering file containing all available processors:
  
  \texttt{Marlin} -x > \texttt{steering.xml}
  
  remove unwanted processors with editor

• Graphical representation of your processor chain:
  
  \texttt{Marlin} -d \texttt{steer.xml} flow.dot
  
  \texttt{dot} -T\texttt{png} flow.dot flow.dot.png
### MarlinGUI

#### List of all Collections Found in LCIO Files

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySimpleFastMCProcessor</td>
<td>SimpleFastMCProcessor</td>
</tr>
<tr>
<td>MyConditionsProcessor</td>
<td>ConditionsProcessor</td>
</tr>
<tr>
<td>MyTestProcessor</td>
<td>TestProcessor</td>
</tr>
<tr>
<td>MyLCIOOutputProcessor</td>
<td>LCIOOutputProcessor</td>
</tr>
</tbody>
</table>

#### Global Section

**Global Section LCIO Files**

CWD: `/home/pclh1-5/kolja`

#### Global Section Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GearXMLFile</td>
<td>gear</td>
</tr>
<tr>
<td>MaxRecordNumber</td>
<td>5001</td>
</tr>
</tbody>
</table>

#### View Options

- Hide Inactive Processors
- Hide Active Processor Errors

#### Active Processors

#### Operations

- Add
- Edit
- Delete
- Deactivate
- Move Up
- Move Down
- Show Cond.

#### Error Description for selected Active Processor

#### Inactive Processors

#### Operations

- Activate
- Edit
- Delete
Writing a Marlin processor

- inherit from a generic Processor class
- implement
  - constructor()
  - init()
  - processRunHeader(LCRunHeader* run)
  - processEvent(LCEvent* evt)
  - end()
- parameters
  - input collections / output collections + collection relations
  - int, double, float, strings …
- advanced logging mechanism (streamlog)
  - level of detail (MESSAGE0-3, WARNING, ERROR)
GeV - Generic Event Viewer

See talk by A. Moll
What the ILC software cannot do at present

• No multithreading:
  • one instance of Marlin has to be started per CPU core
  • works on our CERN Tier2 center with 880 CPU cores with a simple python script
  • ROOT/IO and SIO are not multithreaded!
    – If we implement multithreading: histogram manager + seed manager needed
    – a collection of python scripts for managing these issues
• Cannot write arbitrary objects to LCIO, only LCOObjects
  • very clean data structure!
    – ATLAS and CMS use ROOT but separate data & histograms (ROOT/IO will be implemented as persistency layer within 1 year)
Features of the ILC software

- excellent documentation
  - both the code & usage of the code
- simple and clear structure
- robust, fast and tested in running experiments!
- very modular flexible structure with processors
  - e.g. Tracking is done with several separate processors
- Already adapted to many Belle II needs (PXD/SVD Digitizers & Simulation)
- Belle II can save a lot of manpower by using the existing ILC framework

Very easy for newcomers!
Conclusion

- ILC framework developed and used by ILC/CLIC community
- Modular very flexible structure
  - successfully adapted for Belle II Tracking
  - Geometry of PXD, SVD and CDC implemented
  - Realistic Digitization including clustering is done for PXD and SVD
- Further modifications needed to fully adapt to Belle II
  - PID, ECL and KLM can be added with limited amount of effort
- The ILC framework is running in KEK !!!

Feel free to test it!

/bwf/g67home/kolja/tutorial
Backup
How to start with ILC Frame

• ilc software installed in /bwf/g67home/kolja/ilc
• use bash to source environment script env.software.sh
  • `bash`
  • `. /bwf/g67home/kolja/ilc/env.software.sh`
• All steering files macros and output files can be found in: /bwf/g67home/kolja/tutorial
The Mokka steering file

(mokkaTrkSBelle_CPS1600_SUP10.steer)

/Detector model

/Mokka/init/detectorModel TrkSBelle_CPS1600_SUP10
/Mokka/init/dbHost pcbelle01.mpp.mpg.de

/Mokka/init/modelsDBName Models
/Mokka/init/materialsDBName Materials

/Mokka/init/user belleII
/Mokka/init/dbPasswd belleII

#Geant4 macro file name
/Mokka/init/initialMacroFile pgun2.0GeV80.g4
/Mokka/init/lcioFilename TrkSBelle_CPS1600_SUP10.slcio

# gear output file name
/Mokka/init/MokkaGearFileName gearTrkSBelle_CPS1600_SUP10.xml

/Mokka/init/BatchMode true

no_graphics

Detector model
Geant4 macro
Icio output file
GEAR output file
The Geant4 macro \( \text{pgun2.0GeV80.g4} \)

```
/run/verbose 0
/event/verbose 0
/tracking/verbose 0

# Name of StdHEP file to be used for simulation
# generator/generator
/afs/ipp/home/k/kolja/belle/hepevt/mc03.HEPEvt
/generator/generator particleGun
/gun/position 0 0 0
/gun/direction 0.984808 0 0.173648
/gun/phiSmearing 180 deg
#/gun/thetaSmearing 20
/gun/directionSmearingMode uniform
/gun/energy 2.00 GeV
/gun/particle mu+

# Number of events to be simulated, greater than
# the actual number in HEPEvt file
/run/beamOn 1000
```

- direction \( \theta = 80^\circ \)
- phi-smearing = \( \pm 180^\circ \)
- direction is uniformly generated
- muon particle gun
- optional MonteCarlo simulations
- number of events to sim.
Mokka – the geant4 simulation

• The Mokka steering file \( \text{mokkaTrkBelle\_CPS1600\_SUP10\_steer} \)
  • Detector Model \( \text{TrkS\_Belle\_CPS1600\_SUP10} \)
  • lcio output filename
  • gear output filename
  • geant4 macro filename

• The geant4 macro \( \text{pgun2.0\_GeV80\_g4} \)
  • particle gun
  • (or HEPEvt file containing particles & 4vectors)
  • directions
  • number of events to simulate
LCIO output file and *dumpevent*

- Contains LCCollections:
  - MCParticle (Information about the particles in the simulation)
  - VXDCollection (All simulated hits from PXD and SVD)
  - CDCCollection (All simulated hits in the CDC)
- Contains LCRelations in between individual LCollections

- LCIO can be examined
  - with the program *dumpevent “lciofile” n*
    - where n is the event number (starting from 1)
  - or Marlin, JAS3, GeV or any program see LCIO examples
Marlin processor code 1  
(MarlinVertexResidual)

MarlinVertexResidual aMarlinVertexResidual;

MarlinVertexResidual::MarlinVertexResidual() :
  Processor("MarlinVertexResidual") {
  //Processor description
  _description = "writes out d0 and z0 residuals of the tracks";

  //Register steering parameters
  registerInputCollection(LCIO::TRACK, "TrackCollectionName",
                          "Name of track collection of reconstructed particles",
                          _trackColName, std::string("LDCTracks"));

  registerProcessorParameter("RootOutputFileName",
                           "root file to output the Z0 and D0 residuals",
                           _rootFileName, std::string("residual.root"));
}

create one instance of the processor

processor name

info text for documentation

default values

member variable
Marlin processor code 2
(Easy ROOT integration)

```cpp
void MarlinVertexResidual::init() {
    _nRun = 0, _nEvt = 0;

    _RootFile = new TFile(_rootFileName.c_str(), "RECREATE");
    _RootFile->cd(""");
    _RootTree = new TTree("MarlinTrackTree", "Residuals of the track");
    _RootTree->Branch("D0", &D0, "D0/D");
    _RootTree->Branch("Z0", &Z0, "Z0/D");
}

void MarlinVertexResidual::processEvent(LCEvent* evt) {
    LCCollection * col = 0;
    try {
        col = evt->getCollection(_trackColName.c_str());
    } catch (DataNotAvailableException &e) {}

    Track * trk = dynamic_cast<Track*> (col->getElementAt(0));
    D0 = trk->getD0();
    Z0 = trk->getZ0();
    _RootTree->Fill();
}

void MarlinVertexResidual::end() {
    _RootFile->cd(""");
    _RootFile->Write();
    _RootFile->Close();
}
```
My Marlin steering file 1

```
<marlin>
  <!-- Execute following processors -->
  <execute>
    <processor name="MyMaterialDB"/>
    <processor name="MyVTXDigitizer"/>
    <processor name="MyCDCDigiProcessor"/>
    <processor name="MyLEPTrackingProcessor"/>
    <processor name="MySiliconTracking"/>
    <processor name="MyFullLDCTracking"/>
    <processor name="MyMarlinVertexResidual"/>
  </execute>

  <!-- LCIO input files -->
  <parameter name="LCIOInputFiles"> TrkSBelle_CPS1600_SUP10.slcio </parameter>

  <!-- GEAR input files -->
  <parameter name="GearXMLFile"> gearTrkSBelle_CPS1600_SUP10.xml </parameter>

  <!-- OTHER parameters -->
  <parameter name="Verbosity"> SILENT </parameter>
  <parameter name="MaxRecordNumber" value="1001"/>
  <parameter name="SupressCheck" value="false"/>
  <parameter name="SkipNEvents" value="0"/>
</marlin>
```
My Marlin steering file 2

```
1 <processor name="MyCDCDigiProcessor" type="TPCDigiProcessor">
2 <!--Produces TPC TrackerHit collection from SimTrackerHit collection, smeared in RPhi and Z-->
3 <!--Name of the SimTrackerHit collection-->
4 <parameter name="CollectionName" type="string"> CDCCollection </parameter>
5 <!--Name of the digitized TrackerHit collection-->
6 <parameter name="TPCTrackerHitsCol" type="string"> AllCDCTrackerHits </parameter>
7 </processor>

10 <processor name="MyMarlinVertexResidual" type="MarlinVertexResidual">
11 <parameter name="TrackCollection" type="string" lcioInType="Track"> LDCTracks </parameter>
12 <parameter name="RootOutputFileName" type="string" value="TrkSBelle_CPS1600_SUP10_2.0GeV_80.root"/>
13 </processor>
```