Employing LCIO in Physics Analysis

ECFA Study
Physics and Detectors for a Linear Collider

Jörgen Samson, DESY Hamburg

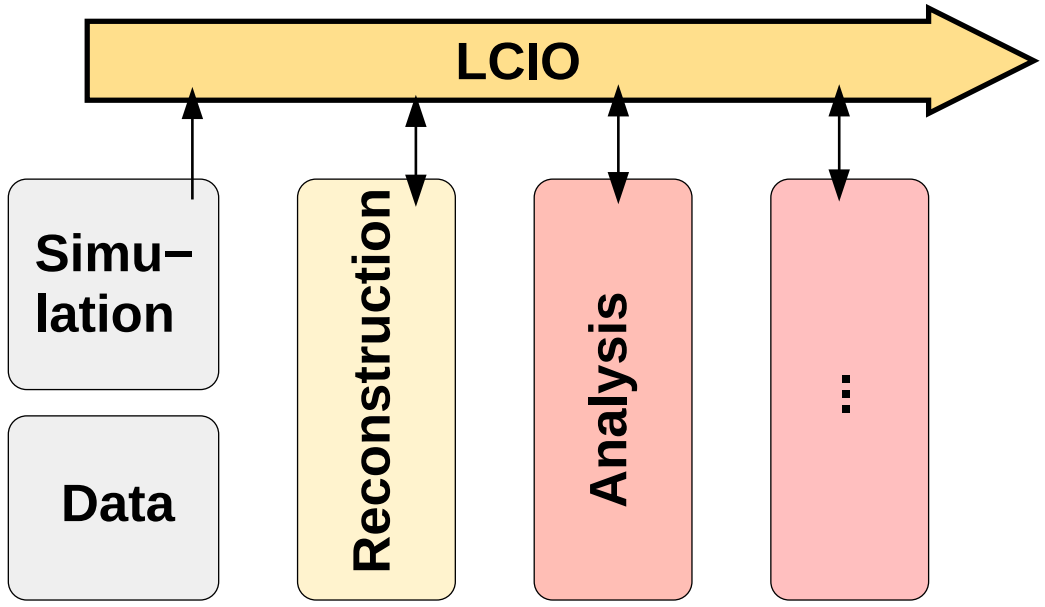
joergen.samson@desy.de

Contents

- Why LCIO for Analysis?
- Some Features of the MARLIN Framework
- The State of Analysis on LCIO
- First Experience
- Performance
- Outlook and Conclusion

What is LCIO?

- from point of view of analysis:
  - LCIO is a data model
  - plus: software interface to access these data
  - plus: persistency framework



Why LCIO for analysis?

- integration and development of analysis tools for LCIO like jet-finder, particle IDs (ZVTOP . . .) must be done
  - to have important tools centrally available
  - to encourage modular structure of analysis code
- this should be done now
  - to test usability of LCIO for analysis and find (small) design problems in LCIO at an early stage

LCIO in use

- LCIO is in use for approx. one year at our group
- up to now mostly for hardware related purposes (calorimeter, TPC) and simulation
- LCIO was not used for physics analysis up to now

$\Rightarrow$ (very) early state of using LCIO for analysis

- with MARLIN a simple data processing framework is already at hand and in use

The MARLIN Framework

First version of a data processing framework (author Frank Gaede)

- MARLIN is implemented in C++ and based on modules
- at startup, a list of modules is dynamically registered with the system
- the modules “act on” events
- the modules are executed serially for each event
- many modules of the same type can appear in this list

The MARLIN Framework

- the registraion of modules (including the order of execution) is controlled by a steering file
- the configuration of each module is also done in the steering file
- MARLIN does not provide a back-end for graphical output or histograming
  $\rightarrow$ you have the free choice (I use ROOT)

```
##########################################
#  Example steering file
##########################################

.begin Global  ------------------------------------
# specify one ore more input files (in one ore more lines)
LCIOInputFiles simjob.slcio

# the active modules that are called in the given order
ActiveModules JetFinderModule
ActiveModules FranksTestModule
ActiveModules OutputModule
.end   ---------------------------------------------
```

the first analysis tools

- there is a LCIO interface for BRAHMS (no full output yet)
- Thorsten Kuhl wrote LCIO output for SIMDET
- multi-algorithm jet-finder as an example for an MARLIN module (wrapper to FORTRAN code, Thorsten Kuhl)

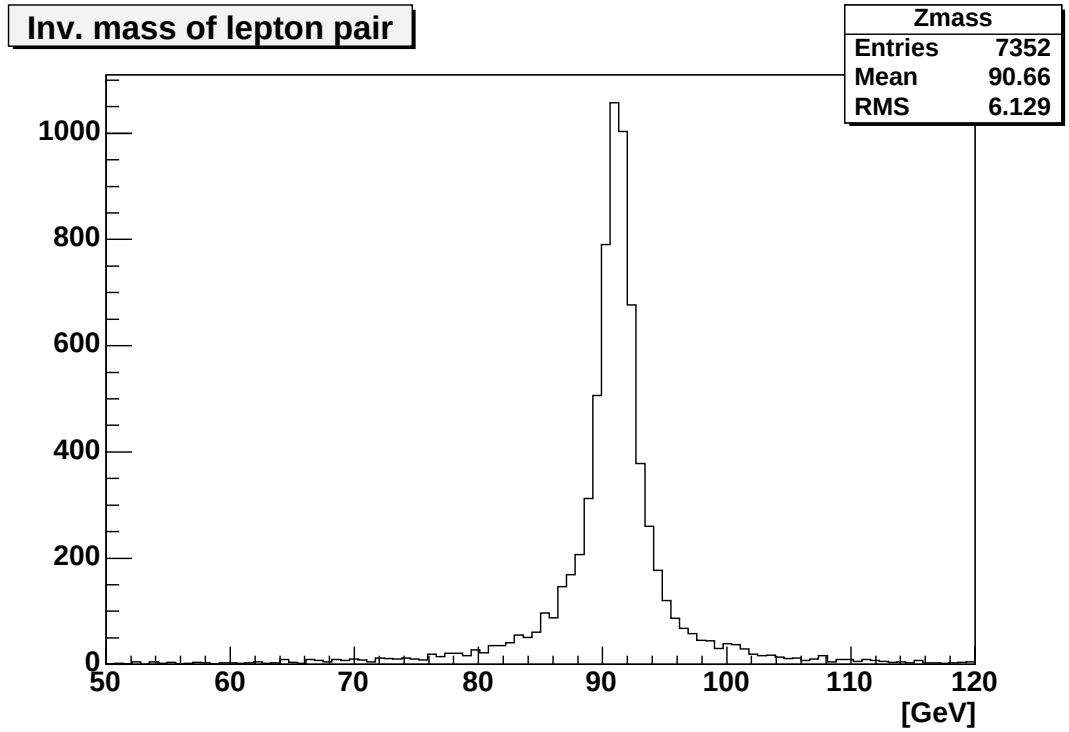
```
.begin JetFinderModule ------------------------------
ModuleType SatoruJetFinderModule

InputCollection ReconstructedParticle
OutputCollection Jets
# DurhamNjet, DurhamYCut, Saturo, Manual
Mode DurhamYCut

# If Mode is Manual-->
NJetRequested  4
YCut 0.01
  ...
Debug 1
.end ------------------------------------------------
```

the first analysis tools

- Lepton finder as first “native LCIO” module for MARLIN
  - finds and identifies isolated high energetic leptons
  - can also easy be used without MARLIN
- Energy cut module
  - sums the energy of all particles in one collection
  - tags the event as “cut off” if sum of energy is too low



Example for a module

```
 1 // Energy Cut Module
 2 // v0.1 Jul 27/04 J. Samson
 3 #ifndef __ENCUTMODULE_H__
 4 #define __ENCUTMODULE_H__
 5
 6 #include "LCIOModule.h"
 7 #include "lcio.h"
 8 using namespace lcio;
 9
10 class EnCutModule : public LCIOModule
11 {
12 public:
13    virtual LCIOModule* newModule(){return new EnCutModule;}
14    EnCutModule();
15
16    virtual void init();
17    virtual void processRunHeader( LCRunHeader* run );
18    virtual void processEvent( LCEvent * evt );
19    virtual void end();
20
21 private:
22    float mCutEnergy;
23    std::string mCollName;
24 };
25 #endif // __ENCUTMODULE_H__
```

Example for a module

```
#include "EnCutModule.h"
using namespace lcio;

EnCutModule aEnCutModule;

void EnCutModule::init(){
  // 1) set cut energy
  mCutEnergy = parameters()->getFloatVal("CutEnergy");
  ...
}

void EnCutModule::processEvent( LCEvent * evt ){
  // **PUNKTE**
  LCCollection* collection=evt->getCollection(mCollName);
  ...
  particle= dynamic_cast<ReconstructedParticle *>
                              collection->getElementAt(i);
  ...
  mEnergy= mEnergy + particle->getEnergy();
  ...
}

void EnCutModule::processRunHeader( LCRunHeader* run){}
void EnCutModule::end(){}
```

remarks

- LCIO does not define the meaning of all data (in particular meaning of bit-fields)
- LCIO authors: ”LCIO should be as general as possible. The user community must find own conventions (which must be documented in the event/run headers)”
- “user community” means “people who want to use the same LCIO file”
  $\rightarrow$ large group
- conventions should be chosen very careful, and well documented - frequent changes can destroy many of the advantages of LCIO

issues

- the linearity of module execution is not flexible enough for analysis
  - at least some abort conditions should be available
  - definition of tree structure for modules?
- conventions to pass information between different analysis modules are needed (attaching information to event, but how exactly?)

performance of LCIO

	File size (example with 10000 events)	read events from disc	typical analysis time
LCIO	100MB	12 ms / event	50-500ms
ROOT	70MB	3 ms / event	50-500ms

- the ROOT IO routines are extremely optimised, LCIO barely optimised
- LCIO is (in principle) independent to the storage format ($\rightarrow$ can be changed)
- IO time is dominated by SIO routines
- optimisation and/or switch to different (faster) IO is conceptually possible

conclusion and outlook

- LCIO provides a fairly generic data model
- this data model also internally connects pieces of information

$\Rightarrow$ lead to easy development of new modules

- specific implementations (e.g. R&D groups . . .) have to be provided by the user community
- performance is (already yet) no argument against LCIO

conclusion and outlook

- the concept of modules allows easy and fast restructuring of an analysis and re-usability of modules in different analysis
- MARLIN is a start and will certainly evolve a lot
- my opinion: software written for LCIO will have a much longer lifetime
- LCIO is still in flux
- advantage will even increase after the LCIO interface becomes stable

references

LCIO on the web

[1] LCIO Homepage (primary source),
http://lcio.desy.de/

[2] Discussion and Bug Report,
http://forum.linearcollider.org; http://bugs.freehep.org

[3] CVS Repository,
:pserver:anonymous@cvs.freehep.org:/cvs/lcio