



LCIO

A persistency framework for LC detector
simulation studies

Frank Gaede, DESY, IT
4th ECFA/DESY LC Workshop
Amsterdam April 1st-4th 2003



Outline

- Introduction
- Data model (brief, see talk in detector performance session)
- Design and Implementation
- Status
- Summary

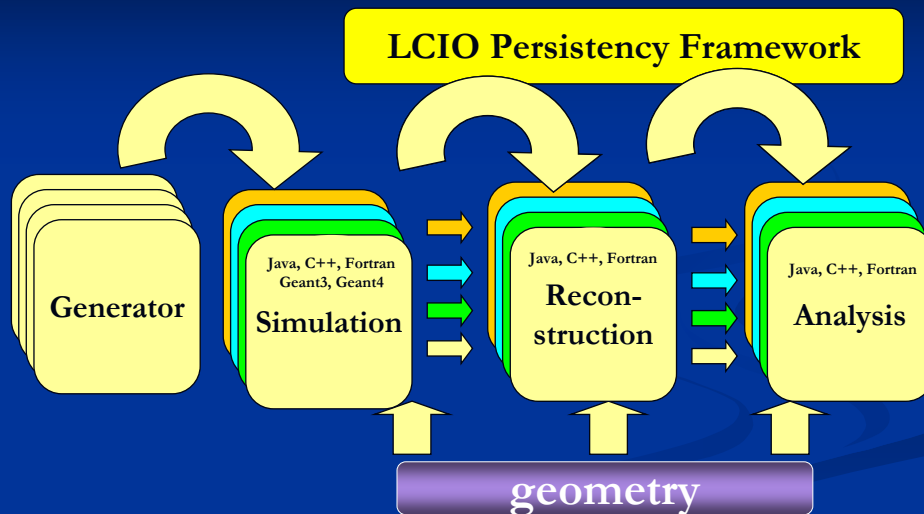


Introduction

- At Prague workshop decided to have **Data format/persistency task force:**
“Define an abstract object persistency layer and a data model for linear collider simulation studies until the Amsterdam workshop.”
- People:
 - Ties Behnke - DESY/SLAC
 - Frank Gaede - DESY
 - Norman Graf - SLAC
 - Tony Johnson - SLAC
 - Paulo Mora de Freitas - IN2P3

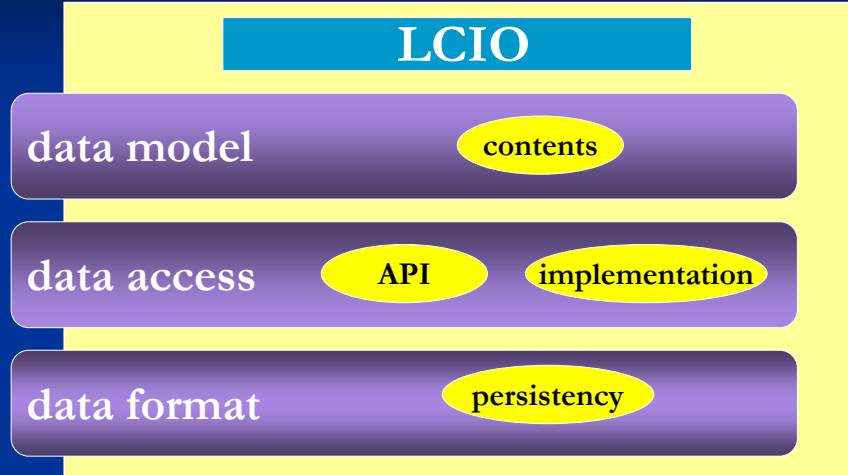


Motivation





The Persistency Framework



Meetings

- Meeting at SLAC 12/09-12/13/02:
(T. Behnke, F. Gaede, N. Graf, T. Johnson)
 - agreement to have common persistency framework in one US group (hep.lcd) and in the European group: **LCIO**
 - agreement on the (first) implementation format
 - first definition of the data model
- Meeting at Ecole Polytechnique 01/14-01/15/03:
(F. Gaede, P. Mora de Freitas, H. Videau, J.-C. Brient)
 - agreement to use LCIO as the output format for the Mokka simulation framework
 - further discussions and refinement of the data model (reconstruction)
- Presentation and discussion of the data model at CERN miniworkshop of detector performance group 25/02/03
- Several phone meetings

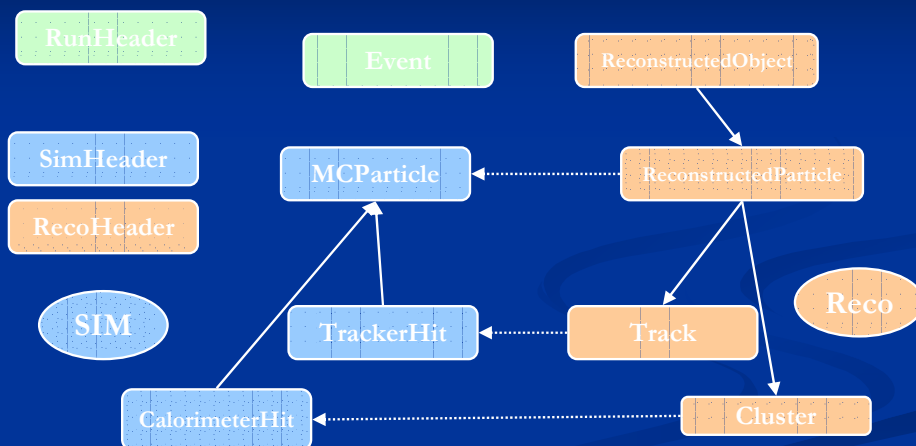


LCIO requirements

- need Java, C++ and f77 (!) implementation
- extendable data model for current and future simulation studies
- user code separated from concrete data format
 - -> want to be flexible for future decisions on persistency
- three major use cases
 - writing data (simulation)
 - reading and updating data (reconstruction)
 - read only access to data (analysis)
- needed a.s.a.p. -> keep it simple !

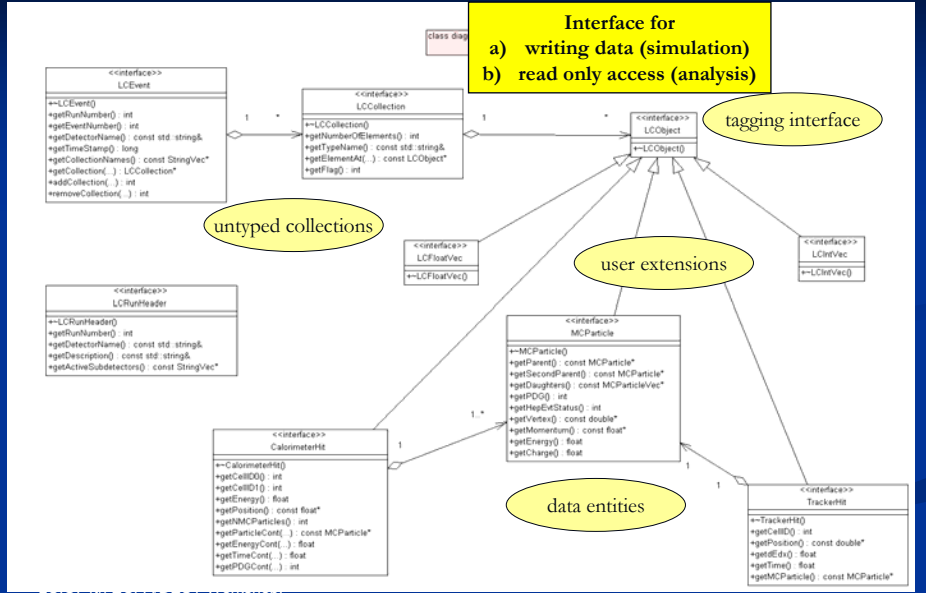


A brief Look at the Data Model:





API – simulation data



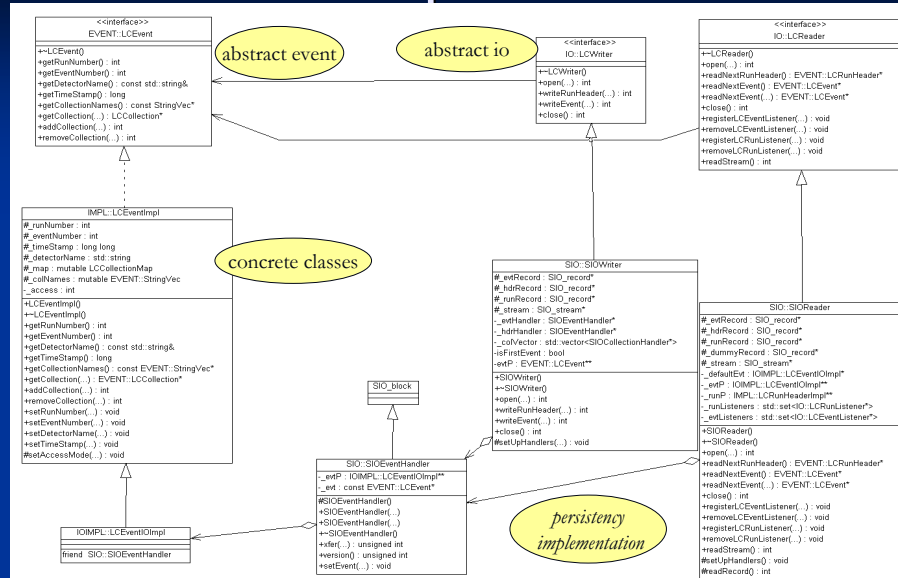
Amsterdam 2003

Frank Gaede, DESY IT

9



API & implementation



LCIO, 4th ECFA/DESY Workshop, Amsterdam 2003

Frank Gaede, DESY IT

10



API definition for Java and C++

- use AID Abstract Interface Definition
 - tool from freehep.org
 - used successfully in the AIDA project
- define interfaces in Java-like language with C++ extensions
 - -> generates files with Java interfaces
 - -> generates C++ header files with pure abstract base classes
 - use `javadoc` for documentation
- independent implementations in Java and C++
 - -> keep Java “pure” i.e. machine independent



LCIO Fortran interface

- Fortran support for
 - legacy software (e.g. BRAHMS reconstruction)
 - non OO-analyses code (“old guys”)
- not a third implementation of the library – use C++-wrapper functions and `cfortran.h` instead:
 - one function for every class member function
 - use integers to store pointers !
 - -> OO-like code in fortran



LCIO f77 example:

The image shows two Emacs windows side-by-side. The left window, titled 'emacs@pcx3340.desy.de <6>', displays Fortran code for an event loop in 'ana.job.f'. The code includes a 'do' loop that reads the next event, gets its run number, event number, and detector name, and prints them. The right window, titled 'emacs@pcx3340.desy.de <7>', displays the corresponding C++ code in 'ana.job.cc'. It uses a 'while' loop with a pointer to an 'LCEvent' object, and uses 'std::cout' for printing. Both windows show the status bar at the bottom of the editor area.

```
----- event loop -----
do 10
  event = lndrreadnextevent( reader )
  if( event.eq.0 ) goto 11

  runnum = levtgetrunnumber( event )
  evtnum = levtgeteventnumber( event )
  detname = levtgetdetectorname( event

  write(*,*) " run: ", runnum
  write(*,*) " evt: ", evtnum
  write(*,*) " det: ", detname

10 continue
11 continue
c ----- end event loop -----

----- ana.job.f (Fortran)--L43--59%-----

// ----- event loop -----
const LCEvent* event ;
while( (event = lcRdr->readNextEvent()) != 0 ){

  int runNum = event->getRunNumber() ;
  int evtNum = event->getEventNumber() ;
  string detName = event->getDetectorName() ;

  std::cout << " run: " << runNum << std::endl ;
  std::cout << " evt: " << evtNum << std::endl ;
  std::cout << " det: " << detName << std::endl ;

}

//----- end event loop -----

----- ana.job.cc (C++ Abbrev)--L16-- 4%-----
```



Persistency Implementation

- use SIO: Simple Input Output
- developed at SLAC for NLC simulation
- already used in hep.lcd framework
- features:
 - on the fly data compression
 - some OO capabilities, e.g. pointers
 - C++ and Java implementation available



Status of LCIO

- C++ implementation available
 - integrated into Mokka simulation framework now
(latest release mokka-01-05 writes TrackerHits in LCIO)
- f77 prototype
 - demonstrating the design
- Java implementation development ongoing
- complete integration into simulation software chains in the next months:
 - US: hep.lcd (Java)
 - Europe: Mokka (C++)/BRAHMS-reco(f77)



LCIO - To Do

- consolidate the API
- make the implementation more robust
 - error handling
 - debugging
- implement reconstruction data model
- add fast skipping mechanism
- get user feedback
 - -> lots of improvements ...



Summary

- LCIO is a persistency framework for linear collider simulation software
- Java, C++ and f77 user interface
- LCIO is currently implemented in simulation frameworks:
 - hep.lcd
 - Mokka/BRAHMS-reco
- > other groups are invited to join
- see LCIO homepage for more details:

<http://www-it.desy.de/physics/projects/simsoft/lcio/index.html>



Intermediate Software Chain

