



LCIO

A persistency framework for linear collider
detector simulation studies

Frank Gaede, DESY, IT
CHEP 2003, San Diego



People

- Ties Behnke - DESY/SLAC
- Frank Gaede - DESY
- Norman Graf - SLAC
- Tony Johnson - SLAC
- Paulo Mora de Freitas - IN2P3

project emerged out of the 'persistency task
force' of the ECFA/DESY workshop

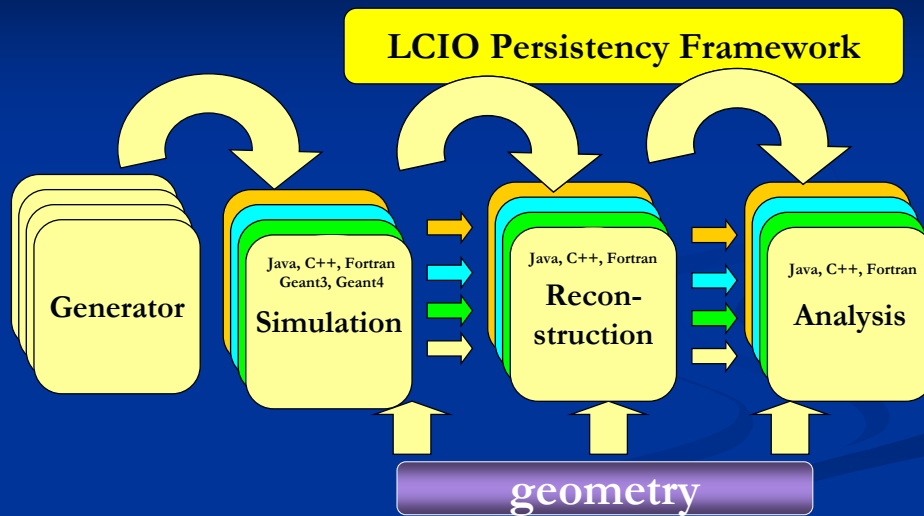


Outline

- Introduction
- Data model
- Software design
- Technical realization
- Data format
- Status
- Summary



Motivation





The Persistency Framework

LCIO

data model

contents

data access

API

implementation

data format

persistency

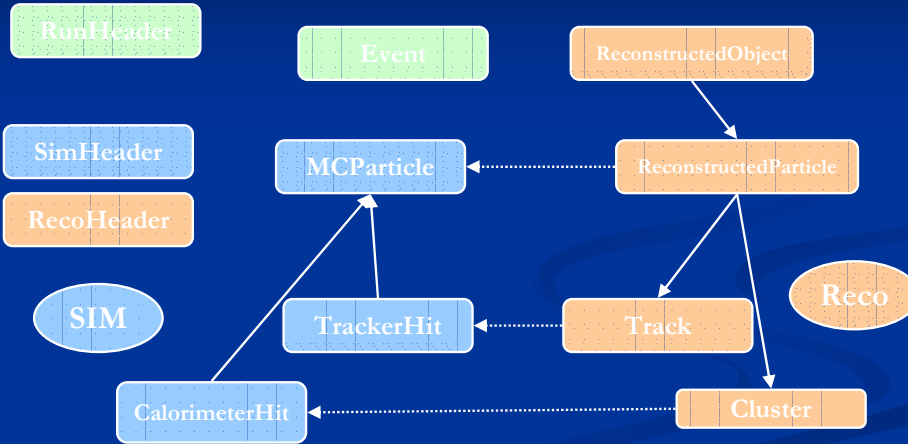


LCIO requirements

- need Java, C++ and f77 (!) implementation
- data model for simulation studies (extendable)
- user code separated from concrete data format
- three major use cases
 - writing data (simulation)
 - reading and updating data (reconstruction)
 - read only access to data (analysis)
- needed asap -> keep it simple !



A quick Look at the Data Model:



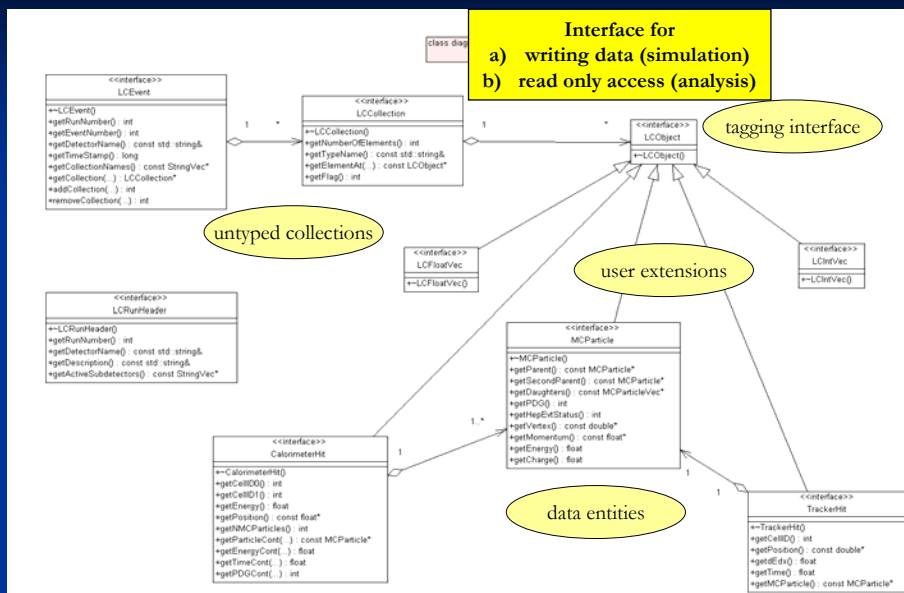
LCIO, CHEP 2003, San Diego

Frank Gaede, DESY

7



API – simulation data



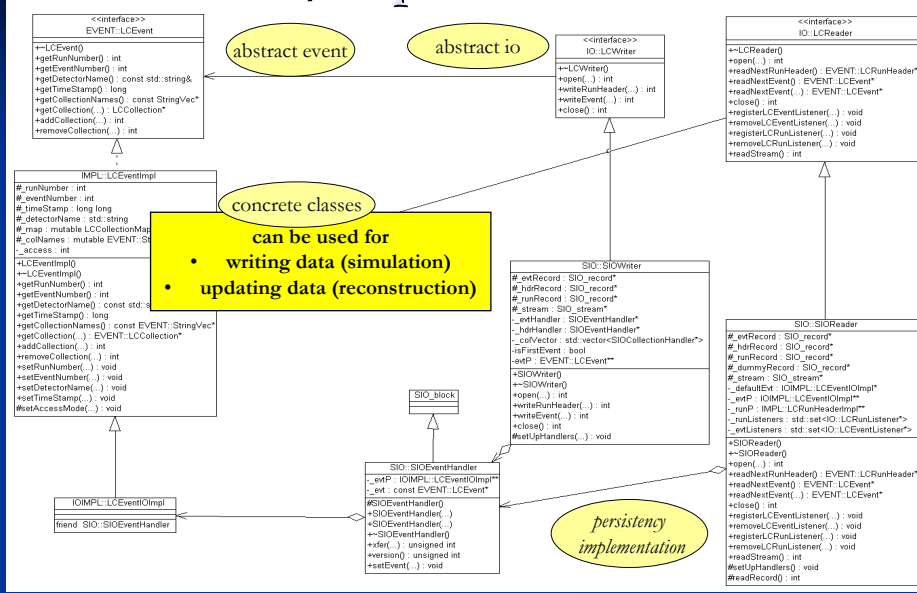
LCIO, CHEP 2003, San Diego

Frank Gaede, DESY

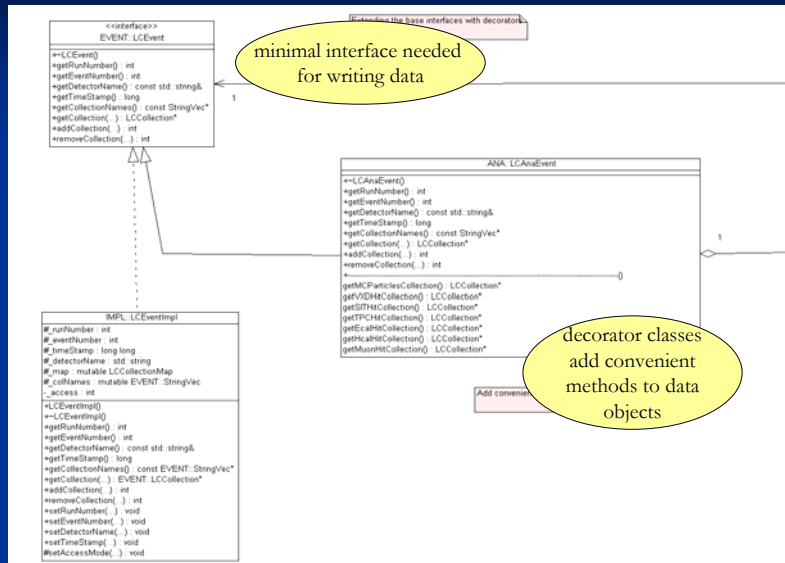
8



API/implementation



Extending the base API





API definition with AID

- 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
- independent implementations in Java and C++
 - -> keep Java “pure” i.e. machine independent



AID example:

```
emacs@pck3340.desy.de ~/7/*
File Edit Options Buffers Tools C Help

#include <string>
#include "LCIO.h"

namespace EVENT {
class LCEvent;
class LCEvent {
public:
    /// Destructor.
    virtual ~LCEvent() { /* nop */; }

    /** Return the run number off this event.
     */
    virtual int getRunNumber() const = 0;

    /** Returns this event's number .
     */
    virtual int getEventNumber() const = 0;

    /** Returns the name of the detector setup used in the
     * simulation.
     */
    virtual const std::string & getDetectorName() const = 0;

    /** Returns the time stamp of the event.
     */
    virtual long getTimeStamp() const = 0;
};
}
LCEvent.h (C Abbrev)--L25--261
```

```
emacs@pck3340.desy.de ~/*
File Edit Options Buffers Tools Java Help

package hep.lcio.event;

/**The main event interface. Has to be implemented by
 * all user event classes that are to be made persistent.
 *
 * Author gaede
 * @version Mar 4, 2003
 * @see LCCollection
 */
public interface LCEvent {

    /** Return the run number off this event.
     */
    public int getRunNumber() const ;

    /** Returns this event's number .
     */
    public int getEventNumber() const ;

    /** Returns the name of the detector setup used in the simulation.
     */
    public const String& getDetectorName() const ;

    /** Returns the time stamp of the event.
     */
    public long getTimeStamp() const ;
}
LCEvent.aid (Java CVS-1.2 Abbrev)--L4--Top
```

```
package hep.lcio.event;

public interface LCEvent {

    /** Return the run number off this event.
     */
    public int getRunNumber();

    /** Returns this event's number .
     */
    public int getEventNumber();

    /** Returns the name of the detector setup
     * used in the simulation.
     */
    public String getDetectorName();

    /** Returns the time stamp of the event.
     */
    public long getTimeStamp();
}
LCEvent.java (Java Abbrev)--L23--27X
```



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 screenshot shows two Emacs windows. The left window, titled 'ana.job.f', displays Fortran code for an event loop. The right window, titled 'ana.job.cc', displays the corresponding C++ code. Both windows show the same logic: reading an event, getting its run number, event number, and detector name, and printing them out.

```
----- 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 -----
```

```
// ----- 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 -----
```



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
- XML files describing the data layout



XML data layout:

```
<block name="MCParticle" major="1" minor="0">
  <data type="int" name="flags">not yesy used</data>
  <data type="int" name="nMC">
  - <repeat count="nMC">
    <data type="ptag" name="this" />
    <data type="pntr" name="parent" />
    <data type="pntr" name="secondParent" />
    <data type="int" name="nDaughter" />
    - <repeat count="nDaughter">
      <data type="pntr" name="daughter" />
    </repeat>
  </repeat>
  <data type="int" name="PDG" />
  <data type="int" name="beginVStatus">= 0 empty line = 1 undecayed particle, stable in the generator = 2 particle decayed in
  the generator = 3 documentation line = 201 stable state, but created in the tracking in the detector = 202 particle
  decayed in the tracking in the detector</data>
  <data type="double[3]" name="vertex">Production vertex</data>
  <data type="float[3]" name="momentum">Momentum at production vertex</data>
  <data type="float" name="energy" />
  <data type="float" name="charge" />
</repeat>
</block>
- <block name="TrackerHit-BlockName" major="1" minor="0">
  <data type="int" name="flags">Bit 31 Barrel = 0, Endcap = 1, rest is detector specific</data>
  <data type="int" name="n">
  - <repeat count="n">
    <data type="int" name="cellID">Detector specific</data>
    <data type="double[3]" name="position" />
    <data type="float" name="detsc" />
    <data type="float" name="time">TODO: Definition Needed (units absolute/relative?)</data>
    <data type="pntr" name="MCParticle" />
  </repeat>
</block>
- <block name="CalorimeterHit-BlockName" major="1" minor="0">
  <data type="int" name="flags">Bit 31 long = 1, short = 0; Bit 30 Barrel = 0, Endcap = 1; BR 29 +z = 1, -z = 0; BR 28 PDG = 1, no
  PDG = 0, rest is detector specific</data>
  <data type="int" name="nHits" />
  - <repeat count="nHits">
    <data type="int" name="cellID">Defines cell ID in detector specific way</data>
    <data type="int" name="cellID1">second word for cell id</data>
    <data type="float" name="energy" />
    - <if condition="(flags&(1<<31)) != 0">
      <data type="double[3]" name="position" />
    </if>
  </repeat>
</block>
```




Status of LCIO

- C++ implementation available
 - will be integrated into Mokka (geant4) simulation framework
- f77 prototype
 - demonstrating the design
- Java implementation developed now
- complete integration into simulation software chains in the next months:
 - US: hep.lcd (Java)
 - Europe: Mokka (C++)/BRAHMS-reco(f77)



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>