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

LCIO Persistency Framework

Generator

Simulation

Reconstruction

Analysis

geometry
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:

- RunHeader
- SimHeader
- RecoHeader
- Event
- MCParticle
- TrackerHit
- CalorimeterHit
- ReconstructedObject
- ReconstructedParticle
- Track
- Cluster
- Reco

API – simulation data

Interface for
a) writing data (simulation)
b) read only access (analysis)
API/implementation

- abstract event
- abstract io
- concrete classes
can be used for
- writing data (simulation)
- updating data (reconstruction)

Extending the base API

- minimal interface needed for writing data
- decorator classes add convenient methods to data objects
API definition with AID

- **AID Abstract Interface Definition**
  - tool from [freehep.org](http://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:**

```java
public interface LCEvent {
    public int getRunNumber();
    public int getEventNumber();
    public String getDetectorSetup();
    public String getSimulatorName();
    public String getSimulatorVersion();
}
```
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:

```fortran
// ---- event loop -----
do 10 event = lReader->nextEvent( reader )
   if( event->nEv() != 0 )

      runnum = event->getRunNumber();
      eventNum = event->getEventNumber();
      detName = event->getDetectorName();
      write(*,*) ' run: ', runnum
      write(*,*) ' ev: ', eventNum
      write(*,*) ' det: ', detName

   10 continue

// ---- 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:

defines the format of the data in the sio files for the two implementations
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: