Introduction to ILC-LDC Simulation and Reconstruction Software

Frank Gaede
DESY
ILC Detector and Physics Workshop, Snowmass
August 14-27, 2005
Outline

- Introduction - overview international software
- Central tools for LDC study
  - LCIO – data model & persistency
  - Simdet – fast simulation
  - Brahms – geant3 full simulation and reconstruction
  - Mokka – geant4 full simulation
  - Marlin – C++ reconstruction framework
  - LCCD - conditions data toolkit
  - GEAR – geometry description
- Summary & Outlook
Detector Concept Study

Need (common?) **Simulation** and **Reconstruction** software to study detector concepts' performance!

Three interregional detector concept studies ongoing

**SiD**: Silicon based concept

**GLD**: Even larger detector concept

**LDC**: Large detector concept

Concepts currently studies differ mainly in **SIZE** and aspect ratio.

Relevant: inner radius of ECAL: defines the overall scale.

ECAL end-view

SiD: Silicon based concept

GLD: Even larger detector concept

LDC: Large detector concept

Main Tracker

EM Calorimeter

HI Calorimeter

Cryostat

Iron Yoke / Murn System
# ILC software packages

<table>
<thead>
<tr>
<th>Description</th>
<th>Detector</th>
<th>Language</th>
<th>IO-Format</th>
<th>Region</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simdet</td>
<td>fast Monte Carlo</td>
<td>TeslaTDR</td>
<td>Fortran</td>
<td>StdHep/LCIO</td>
</tr>
<tr>
<td>SGV</td>
<td>fast Monte Carlo</td>
<td>simple Geometry, flexible</td>
<td>Fortran</td>
<td>None (LCIO)</td>
</tr>
<tr>
<td>Lelaps</td>
<td>fast Monte Carlo</td>
<td>SiD, flexible</td>
<td>C++</td>
<td>SIO, LCIO</td>
</tr>
<tr>
<td>Mokka</td>
<td>full simulation – Geant4</td>
<td>TeslaTDR, LDC, flexible</td>
<td>C++</td>
<td>ASCII, LCIO</td>
</tr>
<tr>
<td>Brahms-Sim</td>
<td>Geant3 – full simulation</td>
<td>TeslaTDR</td>
<td>Fortran</td>
<td>LCIO</td>
</tr>
<tr>
<td>SLIC</td>
<td>full simulation – Geant4</td>
<td>SiD, flexible</td>
<td>C++</td>
<td>LCIO</td>
</tr>
<tr>
<td>LCDG4</td>
<td>full simulation – Geant4</td>
<td>SiD, flexible</td>
<td>C++</td>
<td>SIO, LCIO</td>
</tr>
<tr>
<td>Jupiter</td>
<td>full simulation – Geant4</td>
<td>JLD (GDL)</td>
<td>C++</td>
<td>Root (LCIO)</td>
</tr>
<tr>
<td>Brahms-Reco</td>
<td>reconstruction framework (most complete)</td>
<td>TeslaTDR</td>
<td>Fortran</td>
<td>LCIO</td>
</tr>
<tr>
<td>Marlin</td>
<td>reconstruction and analysis application framework</td>
<td>Flexible</td>
<td>C++</td>
<td>LCIO</td>
</tr>
<tr>
<td>hep.lcd</td>
<td>reconstruction framework</td>
<td>SiD (flexible)</td>
<td>Java</td>
<td>SIO</td>
</tr>
<tr>
<td>org.lcsim</td>
<td>reconstruction framework (under development)</td>
<td>SiD (flexible)</td>
<td>Java</td>
<td>LCIO</td>
</tr>
<tr>
<td>Jupiter-Satellite</td>
<td>reconstruction and analysis</td>
<td>JLD (GDL)</td>
<td>C++</td>
<td>Root</td>
</tr>
<tr>
<td>LCCD</td>
<td>Conditions Data Toolkit</td>
<td>All</td>
<td>C++</td>
<td>MySQL, LCIO</td>
</tr>
<tr>
<td>GEAR</td>
<td>Geometry description</td>
<td>Flexible</td>
<td>C++ (Java?)</td>
<td>XML</td>
</tr>
<tr>
<td>LCIO</td>
<td>Persistency and datamodel</td>
<td>All</td>
<td>Java, C++, Fortran</td>
<td>-</td>
</tr>
<tr>
<td>JAS3/WIRED</td>
<td>Analysis Tool / Event Display</td>
<td>All</td>
<td>Java</td>
<td>xml, stdhep, hep, heprep, LCIO</td>
</tr>
</tbody>
</table>
A Common Software Framework for the ILC?

- a common software framework for the ILC that is flexible and easy to use would be highly desirable
  - (also for detector concept study!)
- but:
  - ILC emerged out of three regional studies
  - all groups have developed their own software as needed for the R&D
  - different languages used: C++, Java, f77
- aim:
  - develop modular software and define interfaces so that packages can coexist/cooperate - (and eventually converge !?)
- common basis: LCIO
Overview of LDC software tools

- **Generator**
  - Mokka
  - Java, C++, Fortran
  - Geant3, Geant4

- **Simulation**
  - Simdet
  - Brahms
  - sim
  - Java, C++, Fortran

- **Reconstruction**
  - Marlin
  - Brahms
  - reco
  - Java, C++, Fortran

- **Analysis**
  - Java, C++, Fortran

- **LCIO Persistency Framework**

- **Geometry/Conditions**
  - GEAR
  - LCCD
LCIO overview

- DESY and SLAC joined project:
  - provide common basis for ILC software
- Features:
  - Java, C++ and f77 (!) API
  - extensible data model for current and future simulation and testbeam studies
  - user code separated from concrete data format
  - no dependency on other frameworks
- simple & lightweight

now de facto standard for ILC software

SW-Architecture

Java API

C++ API

f77 API

JAS/AIDA

root

hbook

common API

LCIO Java implementation

LCIO C++ implementation

*.slcio files (SIO)
LCIO data model

The LCEvent serves as a container of named collections of the various data types in LCIO (LObject subclasses).

Hierarchy of data objects in the event

Event serves as container of untyped collections

Implement new classes as needed for testbeams!
LCIO status  [v01-05]

- changed return values of E, P, m to double for MCParticle and ReconstructedParticle
  - stored are still floats
  - requires some trivial changes (float->double) in code where indicated by the compiler !
- new template UTIL::LCTypedVector<T> for creating std::vector<LCOBJECTTYPE> from LCCollections
  - allows to use iterators and STL algorithms, e.g. std::for_each()
- added first implementations of generic tracker raw data classes (TPC, VTX, SiliconStrip,...)
  - TrackerRawData, TrackerData, TrackerPulse
- files are downward compatible with LCIO 1.4
- bug fixes
- see $LCIO/doc/versions.readme for more
LCIO on the web

home: http://lcio.desy.de
forum: http://forum.linearcollider.org
bugs: http://bugs.freehep.org
LCIO Future Plans

- current data model fairly complete
  - check usability of data model, e.g. track parameters
  - ongoing through development of reconstruction code!
- need to define conventions on how to use LCIO
  - collection names, object types, collections to be present in event
- need to make LCIO more convenient (and efficient)
  - decoding of MCParticle information (parent/daughter, ...)
  - inverse relations (get all tracks for one MCParticle)
  - attach user information to LCOjects
- would like to make C++ version more C++ like, e.g.
  - allow to use STL algorithms (templates in general)

- will have LCIO meeting here at Snowmass
Simulation tools: Simdet, Brahms

**SIMDET**
- parameterized fast Monte Carlo (f77)
- tracks + cov. matrix and clusters
- hard coded geometry: TESLA TDR Detector

**Brahms**
- geant3 full simulation (f77)
- hard coded geometry: TESLA TDR Detector
- full standalone reconstruction part (pflow)
  - tracking based on LEP reconstruction code

for download (cvs web interface )
and more information: [http://www-zeuthen.desy.de/linearCollider](http://www-zeuthen.desy.de/linearCollider)
Simulation tools: Mokka

- geant4 based full detector simulation for the ILC
- developed at LLR-Ecole Polytechnique (P. Mora de Freitas, G. Musat)
- some features:
  - steering files for configuration
  - all geant4 physics lists available
  - writes LCIO
  - reads StdHep / ASCII HepEvt
- Geometry
  - MySQL databases
    - geometry parameters
    - one database per subdetector
  - C++ Geometry Drivers
    - one for each subdetector type (e.g. TPC, HCAL)
    - define material and sensitive detector
  - abstract geometry layer: CGA
- new features: (v05-01)
  - SID detector model
  - scalable detector models
Marlin - Introduction

**Modular Analysis & Reconstruction for the Linear Collider**

- modular C++ application framework for the analysis and reconstruction of LCIO data
- uses LCIO as transient data model
- similar to US org.lcsim Java framework

**Application framework:**
- set of classes that provide the core functionality needed in problem domain and provide hooks (callbacks) for specific user code
- provides main program
- note: most current experiments that use OO (C++) have application frameworks
Marlin – schematic overview

marlin::main

Processor0
Processor1
Processor2
ProcessorN

OutputProcessor

LCEvent
collection0

MyInput0.slcio
MyInput1.slcio
MyInput2.slcio

MyInput.slcio

read and add collections
**Marlin Processor**

- provides main **user callbacks**
- has **own set of input parameters**
  - int, float, string (single and arrays)
  - parameter description
- naturally modularizes the application
- order of processors is defined via steering file:
  - easy to exchange one or several modules w/o recompiling
  - can run the same processor with different parameter set in one job
- processor task can be as simple as creating one histogram or as complex as track finding and fitting in the central tracker

```cpp
marlin::Processor
init()
processRunHeader(LCRunHeader* run)
processEvent(LCEvent* evt)
check(LCEvent* evt)
end()
```

```cpp
UserProcessor
processEvent(LCEvent* evt){
   // your code goes here...
}
```
Marlin features

- **core processors**
  - AIDAProcessor
    - for easy creation of histograms, clouds, ntuples
  - OutputProcessor
    - writes current event or subset thereof
  - MyProcessor
    - simple example – serves as template for user code
  - ConditionsProcessor
    - read conditions transparently with LCCD
  - DataSourceProcessor
    - read non LCIO input, e.g. StdHepReader
  - SimpleFastMCProcessor
    - fast smearing Monte Carlo
    - needs testing

- **fully configurable through steering files:**
  - program flow
  - input parameters
  - processor based and global

- **self-documenting:**
  - MyApplication -l
    will print all available processors with their parameters and example/default values
Marlin - new features I

- **v00-09** (details in `$MARLIN/doc/release.notes`)
- new optional XML steering files
  - based on TinyXml (open source) – included in Marlin
  - same basic structure as old ASCII steering files
- additional features:
  - allows conditional execution of `processEvent()` method based on boolean flags set by processors
  - allows grouping of processors with shared parameters
  - `'Marlin -x > steer.xml' creates example steering file with all available processors
- XML can be used in Marlin for other input files
Marlin – example XML steering

- `<marlin>`
  - `<execute>`
    - `<processor name="MyAIDAProcessor"/>
    - `<processor name="MyEventSelection"/>
  - `<if condition="MyEventSelection">
      - `<group name="Tracking">
          - `<processor name="MyClustering"/>
          - `<processor name="MyPFlow"/>
          - `<processor name="MyLcioOutputProcessor"/>
      </group>
    </if>
  </execute>`
- `<global>`
  - `<parameter name="LCIOInputFiles" value="simjob.slcio"/>
  - `<parameter name="MaxRecordNumber" value="5001"/>
  - `<parameter name="SupressCheck" value="false"/>
- `<processor name="MyLcioOutputProcessor" type="LCIOOutputProcessor"/>
  - `<parameter name="LCIOOutputFile" type="string" value="outputfile.slcio"/>
  - `<parameter name="LCIOWriteMode" type="string" value="WRITE_NEW"/>
- `<group name="Tracking">
  - `<processor name="MyTrackfinder" type="Trackfinder"/>
  - `<processor name="MyTrackfilter" type="Trackfilter"/>
    - `<parameter name="Algorithm" value="DAF"/>
  </group>`
- `<!-- ... -->`
Marlin - new features II

- Exceptions (subclasses of lcio::Exception):
  - uncaught Exceptions terminate program with error message (given in constructor)
  - ParseException
    - thrown if steering has wrong syntax
  - SkipEventException
    - can be thrown by any processor
    - skips processEvent() for all subsequent processors
    - printout at end of program (#evts skipped by processor)
  - StopProcessingException
    - can be thrown by any Processor
    - ends program gracefully by calling all end() methods
- others?
Marlin - new features III

- improved Makefiles
- make it easier to build against an installed version of Marlin:
  - INCLUDE += `. $(MARLIN)/bin/marlin_includes.sh`
  - LIBS += `. $(MARLIN)/bin/marlin_libs.sh`
  - ensures same optional dependencies are used (LCCD, CLHEP, CondDBMySQL,...)
  - USERINCLUDES, USERLIBS to define additional dependencies
- global GNUmakefile:
  - allows to build marlin with several packages of processors
  - links all packages in $MARLIN/packages directory
  - packages need to follow $MARLIN/examples/mymarlin structure
- see workshop DVD
MarlinReco

- Marlin serves as a framework for the distributed development of a full suite of reconstruction algorithms!
  - your input is welcome!

- (almost) complete set of standard reconstruction in Marlin available: MarlinReco (see talk by S. Aplin)
  - first version available on DVD “ILC software for LDC”

- uses first implementation of GEAR geometry description
  - Marlin v00-09-01
Marlin on the web

http://ilcsoft.desy.de/marlin
LCCD Motivation

- fairly complete software chain for simulation and reconstruction for the ILC exists (or is under development)
- can be also used for the simulation of upcoming subdetector testbeam studies
- one important ingredient is missing:

  conditions database

  -> LCCD
LCCD

Linear Collider Conditions Data Toolkit

- handles access to conditions data transparently from
  - conditions database (CondDBMySQL (by Lisbon Atlas group))
  - LCIO files

Conditions Data:
- all data that is needed for analysis/reconstruction besides the actual event data
  - typically has lifetime-validity range longer than one event
    - can change on various timescales, e.g. seconds to years
  - need for versioning (tagging) (changing calibration constants)
  - also 'static' geometry description (channel mapping, positions,...)
LCCD features

- Reading conditions data
  - from conditions database
    - for given tag
  - from simple LCIO file
    - (one set of constants)
  - from LCIO data stream
    - e.g. slow control data
  - from dedicated LCIO-DB file
    - has all constants for given tag
- Writing conditions data
  - as LCGenericObject collection
  - in folder (directory) structure
  - tagging
- Browsing the conditions database
  - through creation of LCIO files
    - vertically (all versions for timestamp)
    - horizontally (all versions for tag)
LCCD on the web

http://ilcsoft.desy.de/lccd
GEAR Overview

**Motivation:**
- need well defined geometry definition that
  - is flexible w.r.t different detector concepts
  - has high level information needed for reconstruction
    (different from detailed local description for simulation !)
  - supports 'plug & play' philosophy of processors implementing different algorithms
  - provides access to material properties (radiation/interaction lengths)

**Idea:**
- define abstract interface (a la LCIO C++ and Java ?)
- concrete implementation based on XML files and CGA
GEAR – Classes

- Subdetector description
  - high level description of detector shape and readout geometry – one class for every subdetector type, e.g.
    - TPC, Ecal, Hcal (MainCalorimeter), FTD, VTX, SIT, ...
  - defines required attributes - as detailed as necessary but as abstract as possible
  - allows to add additional named attributes
  - use XML files

- Material properties
  - point properties (density, material, radlen,...)
  - distance properties integrated along (straight!?) path
  - use Mokka-CGA interface to geant4 geometry
GEAR – TPC description

holds all subdetector classes

- GEAR
  - TPCTpc
    - GearMgr
      + GearMgr()
      + getGearParameters(key : const std::string&) : const GearParameters&
      + getTPCParameters() : const TPCParameters&

- GearParameters
  + ~GearParameters()
  + getIntVal(key : const std::string&) : int
  + getDoubleVal(key : const std::string&) : double
  + getStringVal(key : const std::string&) : const std::string&
  + getUIntVal(key : const std::string&) : const std::vector<int>&
  + getDoubleVal(key : const std::string&) : const std::vector<double>&
  + getStringVal(key : const std::string&) : const std::vector<std::string>&

- PadRowLayout2D
  - CARTESIAN : const int
  - POLAR : const int
  - RECTANGLE : const int
  - DIAMOND : const int
  - HEXAGON : const int
  - CHEVRON : const int
  - PadRowLayout2D()
  + getPadLayoutType() : int
  + getPadShape() : int
  + getNPad() : int
  + getNR() : int
  + getRowHeightInRowNumber(int) : double
  + getPadWidth(padIndex : int) : double
  + getPadCenter(padIndex : int) : Point2D
  + getPadHeight(padIndex : int) : double
  + getPadinRow(rowNumber : int) : const std::vector<int>&
  + getPadExt() : const std::vector<double>&
  + getRowNumber(padIndex : int) : int
  + getPadNumber(padIndex : int) : int
  + getPadNumber(rowNum : int, padNum : int) : int
  + getNearestPad(c0 : double, c1 : double) : int
  + getLeftNeighbour(padIndex : int) : int
  + getRightNeighbour(padIndex : int) : int
  + insidePad(c0 : double, c1 : double, padIndex : int) : bool
  + isInsidePad(c0 : double, c1 : double) : bool

- FixedPadSizeDiskLayout
  - implementationsForDiskLayout() : int

TPC specific parameters

- TPCParameters
  - TPCParameters()
  + getPadLayout() getMaximumDriftLength()
  + getDriftVelocity()

based on discussion with TPC experts at LCWS 2005

implementation for disk with pad rings

can be also used for FTD, CaloEndcap,...
GEAR – material properties

### GearDistanceProperties

- `GearDistanceProperties()`
- `getMaterialNames(p0 : const Point3D&, p1 : const Point3D&) : const std::vector< std::string >&`
- `getMaterialThicknesses(p0 : const Point3D&, p1 : const Point3D&) : const std::vector< double >&`
- `getNRadlen(p0 : const Point3D&, p1 : const Point3D&) : double`
- `getNIntlen(p0 : const Point3D&, p1 : const Point3D&) : double`
- `getBdL(pos : const Point3D&) : double`
- `getEdL(pos : const Point3D&) : double`

### GearPointProperties

- `GearPointProperties()`
- `getCellID(pos : const Point3D&) : int`
- `getMaterialName(pos : const Point3D&) : const std::string&`
- `getDensity(pos : const Point3D&) : double`
- `getTemperature(pos : const Point3D&) : double`
- `getPressure(pos : const Point3D&) : double`
- `getRadlen(pos : const Point3D&) : double`
- `getIntlen(pos : const Point3D&) : double`
- `getLocalPosition(pos : const Point3D&) : Point3D`
- `getB(pos : const Point3D&) : double`
- `getE(pos : const Point3D&) : double`
- `getListOfLogicalVolumes(pos : const Point3D&) : std::vector< std::string >`
- `getListOfPhysicalVolumes(pos : const Point3D&) : std::vector< std::string >`
- `getRegion(pos : const Point3D&) : std::string`
- `isTracker(pos : const Point3D&) : bool`
- `isCalorimeter(pos : const Point3D&) : bool`

integrated along path:
- straight line or
- true path in B-Field?

properties at point from geant4 (CGA)

based on discussions at Argonne Simulation Meeting 2004
GEAR status and outlook

- first mini implementation with XML files for current MarlinReco: TPC, Ecal, Hcal
- XML format 'compatible' with US compact description
- integrated in Marlin v00-09-01 (see DVD)
- soon to be released!

Plans:

- have discussions at snowmass with other subdetector groups about abstract interface needed
- see if other groups are interested in collaborating !?
- provide implementation of material properties based on CGA/Mokka
- investigate option of creating GEAR XML files in Mokka geometry drivers:
  - the information is there
  - have only one source of geometry
LDC simulation framework

Simulation

- MOKKA
  - CGA
    - C++ Drivers
  - Geant4
- MySQL

Reconstruction-Analysis

- MARLIN
  - Processors
  - MarlinUtil
    - CLHEP, gsl,..
- LCCD
  - CondDBMySQL

GEAR
Summary & Outlook

- A fairly complete OO-software framework exists for the LDC study based on Mokka, Marlin, LCIO, LCCD, and GEAR.

- Ready to start using it for detector concept study!

Current version of this software can be found on the DVD “ILC software for LDC”.

- Has been used to produce LDC events on “International ILC Event DVD” on the DESY – computing Grid!

- LCIO files for SID, LDC, GLC of Zpole, Zh and ttbar events

- Also at http://www-flc.desy.de/snowmassdatadvd

To Do:

- Investigate interoperability with other frameworks
  - Apply software to other concepts
  - Exchange ideas and collaborate
  - Improve software ...

All software (will be) available at portal:

http://www-flc.desy.de/ILCsoft