Simulation and Reconstruction Software for the ILC

Frank Gaede, DESY
ACAT05, DESY Zeuthen
May 25, 2005
Outline

- ILC - brief overview
- Overview international software
- Software developed at DESY:
  - LCIO – data model & persistency
  - MARLIN – C++ reconstruction framework
  - LCCD - conditions data toolkit
- Summary & Outlook
ILC Overview

- future linear e+/e- collider (l~30km)
  - super-conducting RF technology
  - stage 1: E=200→500 GeV
  - stage 2: upgrade E~1TeV
- options: gama/gamma, gamma/e-, e-/e-, Giga-Z, 2 IRs for 2 experiments?
- rough timeline (planned):
  - (2005) Accelerator CDR
  - (2007) Accelerator TDR – Detector CDRs
  - (2008) LC site selection
  - (2009) Global lab selects experiments
  - (?) Start construction
  - (??) Data
  - -> operating simultaneously with LHC!
- currently ongoing: R&D, detector concept studies
Detector Concept Study

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

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

Relevant: inner radius of ECAL: defines the overall scale

SiD: Silicon based concept

GLD: even larger detector concept

LDC: large detector concept

three interregional detector concept studies ongoing
Reconstruction @ the ILC

- general ILC detector features:
  - precision tracking
  - precision vertexing
  - high granularity in calorimeters
    - (Ecal ~1cm, Hcal ~1-5cm)
- important: very high jet-mass resolution \( \sim 30\%/\sqrt{E/\text{GeV}} \)

Particle Flow

- reconstruct all single particles
- use tracker for charged particles
- use Ecal for photons
- use Hcal for neutral hadrons

dominant contribution (\(E<50\) GeV):
- Hcal resolution
- confusion term

\[
\sigma_{E_{\text{jet}}}^2 = \epsilon_{\text{trk}}^2 \sum_i E_{\text{trk},i}^4 + \epsilon_{\text{Ecal}}^2 E_{\text{Ecal}} + \epsilon_{\text{Hcal}}^2 E_{\text{Hcal}} + \sigma_{\text{confusion}}^2
\]

\[
\epsilon_{\text{trk}} = \delta (1/p) \approx 5 \cdot 10^{-5}, \quad \epsilon_{\text{Ecal}} = \frac{\delta E}{\sqrt{E}} \approx 0.1, \quad \epsilon_{\text{Hcal}} \approx 0.5
\]
Imaging calorimeter & ParticleFlow

Need software tools to improve clustering & pflow:
- detailed hadronic shower simulation (Geant4)
- framework for developing and comparing pflow algorithms

Note: precision tracking and vertexing also very challenging and important!
## 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>EU</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>ASCI, LCIO</td>
</tr>
<tr>
<td>Brahms-Sim</td>
<td>Geant3 – full simulation</td>
<td>TeslaTDR</td>
<td>Fortran</td>
<td>EU</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>EU</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-Satelite</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>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, LCIO,</td>
</tr>
</tbody>
</table>
ILC Simulation Frameworks (Geant4)

- Geant4, StdHep and LCIO are common feature
- Each trying to be generic with different approach
different ways to define geometries

Geant4, StdHep and LCIO are common feature
- Each trying to be generic with different approach
different ways to define geometries

**Parameter List**
- **MySQL+CGA**
- **XML**
- **LCDD**
- **CGD**

**Jupiter**
- **Mokka**
- **LCDG4**
- **SLIC**

**root**
- **sio**

**LCIO**: SimTrackerHits/SimCalorimeterHits

(A.Miyamoto)
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!
- 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
- example: LCIO
Motivation for LCIO

before:

Generator

Java, C++, Fortran
Geant3, Geant4

Simulation

Geometry

Reconstruction

Java, C++, Fortran

Analysis

Java, C++, Fortran
Motivation for LCIO

after:

LCIO Persistency Framework

Generator

Simulation

Java, C++, Fortran
Geant3, Geant4

Reconstruction

Java, C++, Fortran

Analysis

Java, C++, Fortran

geometry
LCIO project overview

DESY and SLAC joined project:
- provide common basis for ILC software
- started end of 2002

Requirements
- need Java, C++ and f77 (!) API
- extensible data model for current and future simulation and testbeam studies
- user code separated from concrete data format
- easy to adapt LCIO in existing applications
- no dependency on other frameworks

-> keep it simple & lightweight
LCIO Building Blocks

- **data model**: contents
- **data handling**: API, implementation
- **data format**: persistency
LCIO SW-Architecture

- **JAS/AIDA**
- **root**
- **hbook**

**Java API**
- LCIO Java implementation

**C++ API**
- LCIO C++ implementation

**f77 API**

**common API**
- generated from one source using AIDA

***.slcio files (SIO)**
- compressed records, pointer retrieval
LCIO Class Design

abstract event

abstract io

concrete classes

persistency implementation
LCIO data model

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

Implement new classes as needed for testbeams!
LCIO data model defines the object needed for ILC simulation studies, but

- users want additional information in files for specific studies
- can't create new classes within LCIO for all requests and purposes
- need generic user class:
  **LCGenericObject**
  - almost arbitrary data objects
  - typically access provided through user subclass - but not needed:
  - has description string for reading the data without need to have access to data dictionary (library)
LCIO on the web

home:  http://lcio.desy.de
forum:  http://forum.linearcollider.org
bugs:  http://bugs.freehep.org
JAS3 and LCIO

- for all LCIO files:
  - generic Event-Display
  - file browser
  - analysis modules
LCIO - Summary

- LCIO provides a common data model and file format for ILC studies
- has become a defacto standard for ILC software packages
- provides Java, C++ and Fortran interface
- could serve as a basis for a common ILC software framework!

LCIO can also be used as transient data model in analysis and reconstruction software -> MARLIN

<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>EU</td>
</tr>
<tr>
<td>SGV</td>
<td>fast Monte Carlo</td>
<td>simple Geometry, flex</td>
<td>Fortran</td>
<td>EU</td>
</tr>
<tr>
<td>Lelaps</td>
<td>fast Monte Carlo</td>
<td>SID, flexible</td>
<td>C++</td>
<td>EU</td>
</tr>
<tr>
<td>Mokka</td>
<td>full simulation – Geant4</td>
<td>TeslaTDR, LDC, flexible</td>
<td>C++</td>
<td>EU</td>
</tr>
<tr>
<td>Brahms-Sim</td>
<td>Geant3 – full simulation</td>
<td>TeslaTDR</td>
<td>Fortran</td>
<td>EU</td>
</tr>
<tr>
<td>SLIC</td>
<td>full simulation – Geant4</td>
<td>SID, flexible</td>
<td>C++</td>
<td>EU</td>
</tr>
<tr>
<td>LCDG4</td>
<td>full simulation – Geant4</td>
<td>SID, flexible</td>
<td>C++</td>
<td>US</td>
</tr>
<tr>
<td>Jupiter</td>
<td>full simulation – Geant4</td>
<td>JLD (GDL)</td>
<td>C++</td>
<td>EU</td>
</tr>
<tr>
<td>Brahms-Reco</td>
<td>reconstruction framework (most complete)</td>
<td>TeslaTDR</td>
<td>Fortran</td>
<td>EU</td>
</tr>
<tr>
<td>Marlin</td>
<td>reconstruction and analysis application framework</td>
<td>Flexible</td>
<td>C++</td>
<td>EU</td>
</tr>
<tr>
<td>hep.lcd</td>
<td>reconstruction framework</td>
<td>SID (flexible)</td>
<td>Java</td>
<td>US</td>
</tr>
<tr>
<td>org.lcsim</td>
<td>reconstruction framework (under development)</td>
<td>SID (flexible)</td>
<td>Java</td>
<td>US</td>
</tr>
<tr>
<td>Jupiter-Satellite</td>
<td>reconstruction and analysis</td>
<td>JLD (GDL)</td>
<td>C++</td>
<td>AS</td>
</tr>
<tr>
<td>LCCD</td>
<td>Conditions Data Toolkit</td>
<td>All</td>
<td>C++</td>
<td>EU</td>
</tr>
<tr>
<td>LCIO</td>
<td>Persistency and datamodel</td>
<td>All</td>
<td>Java, C++, Fortran</td>
<td>AS,EU,US</td>
</tr>
<tr>
<td>JAS3/WIRED</td>
<td>Analysis Tool / Event Display</td>
<td>All</td>
<td>Java</td>
<td>US,EU</td>
</tr>
</tbody>
</table>

LCIO can also be used as transient data model in analysis and reconstruction software -> MARLIN
Marlin - Motivation

- currently the most complete ILC reconstruction software implementing the pflow algorithm is **Brahms**
  - written in Fortran
  - code recycled from LEP and other experiments
  - based on geant3
  - hard coded geometry definition
  - inflexible and hard to maintain/extend

- so there is need for something new which
  - is written in OO-language
  - is flexible in terms of the detector geometry
  - is easy to use and maintain
  - allows for collaborative development and exchange of code and algorithms
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

LCEvent

collection0

read and add collections

MyInput.slcio

MyInput0.slcio

MyInput1.slcio

MyInput2.slcio

Processor0

Processor1

Processor2

…

ProcessorN

OutputProcessor
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
  - **ConditionsProcessor**
    - read conditions transparently with LCCD
  - **OverlayProcessor**
    - event mixing (under development)
  - **SimpleFastMCProcessor**
    - fast smearing Monte Carlo (under development)
  - **MyProcessor**
    - simple example – serves as template for user code

- **fully configurable through steering files**
- **self-documenting:**
  - MyApplication -l
    will print all available processors with their parameters and example/default values

- **not yet:**
  - logging manager
  - exception handling
  - program flow control
  - ...

Marlin users

- Marlin serves as a framework for the distributed development of a full suite of reconstruction algorithms!
- **aim: have (at least one) complete set for standard reconstruction in C++ soon!**

ongoing activities:

- **Reconstruction software**
  - wrapper for Brahms-Tracking code (S.Aplin)
  - clustering and pflow (A.Raspereza,V.Morgunov)
  - clustering algorithms (Ch. Ainsley, G. Mavromanolakis)

- **Analysis software**
  - LCLeptonFinder (J.Samson)
  - JetFinder (Th.Kuhl)
  - ThrustFinder (Th. Kraemer)

- **CALICE testbeam software**
  - DigiSim (G.Lima)
  - Ganging and Calibration (R.Poeschl)
  - probably others ...
Marlin on the web

http://ilcsoft.desy.de/marlin

Releases

v00-08 has been released and is available for download. Marlin can now optionally be linked against LCCD to provide easy access to conditions data. documentation has been improved.

Download

All tagged versions and the current HEAD of the repository can be downloaded from the

Documentation

Current API documentation.

Talks

LCIO & Marlin (pdf) - talk given at the DESY Simulation WS 2004.

Last modified: Fri Mar 11 16:01:59 2005

by Frank.Gaede@desy.de

Installation

The installation of Marlin is described in the README.

Running Marlin

After having installed Marlin you have to write your own marlin:Processor(s) subclass that performs the computation. This is fairly straightforward and Marlin provides an example in /examples/marlin that can serve as a template for your own projects.

Note: there is no need to write a main program as this is provided by Marlin. Existing Processors are automatically registered with Marlin provided one instance exists in the library as described in the README.

Steering files

Frank Gaede, DESY, ACAT05, DESY Zeuthen, May 25, 2005
Marlin - Summary & Outlook

- modular C++ application framework for the analysis and reconstruction of LCIO data
- ongoing development of reconstruction processors with the aim to have (at least one) complete set for standard reconstruction in C++ soon!
  - until then: reconstruction workhorse still Brahms (f77)

**To Do:**

- **geometry description for reconstruction** – ongoing:
  - have abstract geometry API ala LCIO that can be used in C++ and Java
  - -> need agreement in ILC community
  - also a good idea for Marlin alone ...

- investigate options for interoperability with US framework, i.e.
  **Java-C++ interfacing**
  - technically feasible, but not so straight forward how to do this properly!

both features needed for a common ILC framework!
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

LCCD is a toolkit that enables users to transparently read conditions data from LCIO files or a conditions database. See the API documentation for more. LCCD is still under development - so please test before you use it for production.

Releases
v00-02 has been released and is available for download (requires LCIO v01-04).

Download
All tagged versions and the current HEAD of the repository can be downloaded from the

Documentation
Current (v00-02) API documentation.

Talks
LCCD Proposal [pdf] - talk given in the software meeting @ Desy.

Last modified: Fri Mar 11 15:58:08 2005
by Frank.Gaede@desy.de

http://ilcsoft.desy.de/lccd
CALICE testbeam

- testbeams for CALICE prototypes (Ecal, Hcal, Tailcatcher)
  - DESY 2005:
    - e- 1-6 GeV
  - FNAL 2006/2007
    - mu, pi, p 1-100 GeV
- $O(10^8)$ events
- $n*O(10^8)$ Monte Carlo events
- $\sim 20000$ channels

- detector R&D
  - test DAQ
  - test detector concepts
- software
  - hadronic physics in Geant4
  - **test complete software chain**!
Calice testbeam software

Data Processing Scheme

Calibration/Analysis Steps use LCIO as backbone

- Raw Data → Decoding → Raw LCIO (blocks of integers?)
- Filtering → Mapping → LCIO Raw hits (transient)
- Calibration, Alignment → LCIO (for analysis)

- Mokka LCIO
- True Energy
- Anti-Calibration digitization

SimCalHits

Realization of Scheme?

Marlin Processors
Summary & Outlook

- ILC software development is a very active field, e.g.
  - Mokka, Marlin, LCIO, LCCD, ...
  - currently a full C++ reconstruction framework developed within Marlin

- next topics:
  - abstract geometry interface
  - breaking the Java-C++ - language barrier

Holy grail: have an internationally used common software framework for the ILC that is used to support R&D and the detector concept studies!

There is work to do – but we are in a reasonably good shape given the ILC timescale ...