“A Comprehensive Java-based Simulation Environment for Particle Physics”

Tony Johnson
SLAC
September 2006
What will I really talk about?

- This is actually one of a series of talks/tutorials at Fermilab this week
  - Computing Techniques Seminar
    - Software Framework for ILC detector simulation
      - (Java, C++, XML)
      - Emphasis on Computing Techniques
  - ILC R&D seminar (Wednesday, 11am)
    - Norman Graf will discuss the same framework, but with more emphasis on detector technology studies and related physics
  - Hands on Tutorials
    - This afternoon, and tomorrow morning.
      - We will show how to install and use this software.
Contents

- Goals
- LCIO – Common IO for ILC detector studies
- SLIC – Simulation Package
- Geometry Description
- org.lcsim – Reconstruction and Analysis Package
  - Using org.lcsim with JAS3
  - Using org.lcsim with WIRED4
  - Using org.lcsim with “The Grid”
- How to get involved
Goals

- Enable full studies of ILC physics to optimize detector design and eventual physics output
  - Use realistic detector geometries
  - Full simulation (in combination with fast parameterized MCs)
  - Full reconstruction
    - Simulate benchmark physics processes on different full detector designs.
    - Encourage development of realistic analysis algorithms
    - See how these algorithms work with full detector simulations
- Facilitate contribution from physicists in different locations with various amounts of time available (normally not much!)
  - Software should be easy to install, learn, use
    - Goal is to allow software to be installed from CD with no external dependencies
    - Support via web based forums, tutorials, meetings.
Goals

- Use standard data formats, when possible.
  - Interoperate with other ILC software where possible

- Geometry, “Detector Concept” independent.
# ILC software packages (circa 2004)

<table>
<thead>
<tr>
<th>Description</th>
<th>Detector</th>
<th>Language IO-Format Region</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Simdet</strong></td>
<td>fast Monte Carlo</td>
<td>TeslaTDR</td>
</tr>
<tr>
<td><strong>SGV</strong></td>
<td>fast Monte Carlo</td>
<td>simple Geometry, flexible</td>
</tr>
<tr>
<td><strong>Lelaps</strong></td>
<td>fast Monte Carlo</td>
<td>SiD, flexible</td>
</tr>
<tr>
<td><strong>Mokka</strong></td>
<td>full simulation</td>
<td>Geant4</td>
</tr>
<tr>
<td><strong>Brahms-Sim</strong></td>
<td>full simulation</td>
<td>Geant4</td>
</tr>
<tr>
<td><strong>SLIC</strong></td>
<td>full simulation</td>
<td>SiD, flexible</td>
</tr>
<tr>
<td><strong>LCDG4</strong></td>
<td>full simulation</td>
<td>SiD, flexible</td>
</tr>
<tr>
<td><strong>Jupiter</strong></td>
<td>full simulation</td>
<td>SiD, flexible</td>
</tr>
<tr>
<td><strong>Brahms-Reco</strong></td>
<td>reconstruction framework</td>
<td>TeslaTDR</td>
</tr>
<tr>
<td><strong>Marlin</strong></td>
<td>reconstruction and analysis</td>
<td>Flexible</td>
</tr>
<tr>
<td><strong>hep.lcd</strong></td>
<td>reconstruction framework</td>
<td>SiD (flexible)</td>
</tr>
<tr>
<td><strong>org.lcsim</strong></td>
<td>reconstruction framework (under development)</td>
<td>SiD (flexible)</td>
</tr>
<tr>
<td><strong>Jupiter-Satelite</strong></td>
<td>reconstruction and analysis</td>
<td>JLD (GDL)</td>
</tr>
<tr>
<td><strong>LCCD</strong></td>
<td>Conditions Data Toolkit</td>
<td>All</td>
</tr>
<tr>
<td><strong>GEAR</strong></td>
<td>Geometry description</td>
<td>Flexible (Java?)</td>
</tr>
<tr>
<td><strong>LCIO</strong></td>
<td>Persistency and datamodel</td>
<td>All</td>
</tr>
<tr>
<td><strong>JAS3/WIRED</strong></td>
<td>Analysis Tool / Event Display</td>
<td>All</td>
</tr>
</tbody>
</table>
LCIO

- Object model and persistency
  - Events
    - Monte Carlo
    - Raw
    - Event and run metadata
  - Reconstruction
  - Parameters, relations, attributes, arrays, generic objects, ...
- All the ILC simulators write LCIO
  - Enables cross-checks between data from different simulators
  - Read/write LCIO from
    - Fast MC / Full Simulation
    - Different detectors
    - Different reconstruction tools
Overview: “ALCPG” Framework

- StdHep Events
- SLIC
- LCDD XML
- Compact XML
- LCIO Events
- org.lcsim
- Geom Converter
- JAS3
- Conditions
- HepRep XML
- AIDA
- Software Package
- Data Format
- User Analysis Drivers
- WIRED4

The diagram illustrates the components and interactions within the ALCPG framework.
SLIC – Simulator for LInear Collider

- StdHep
- Physics
- Events
- LCDD
- XML
- Geometry
- SLIC
- Geant4
- Simulator
- Geant4/SLIC Commands + macros
- reads
- reads
- “One .exe to run them all”

- LCDD
- XML
- Geometry
- writes
- LCIO Output File
Prototyping Detectors using Geant4

- Traditional way to build Geant4 geometries is by writing C++ code
  - Positions sizes either hardwired into code or read from database or conditions framework
    - Advantages:
      - Most efficient way to define G4 geometries
      - Full access to G4 geometry features
      - Ideal for “fixed” geometry (e.g. LHC detector)
    - Disadvantages
      - Changes in geometry typically require recoding of C++
      - Access to geometry from other tools (visualization, reconstruction, analysis) difficult
      - Not ideal for prototyping detectors, test beams, thought experiments etc.

- Alternative approach is to use XML to define G4 geometries
  - Human readable and editable
  - Quick development cycle
    - no recompilation for geometry changes
    - properties easily “tweaked”
  - Portable standard: easy to import/export/exchange data
  - High quality, standardized tools in C++ and Java, Python
  - Self-descriptive with schemas (XSD)
    - validating parser quickly identifies errors
  - Natural representation of structured hierarchies, i.e. detector geometries
  - No database to install or access at runtime
    - Easy to run on Grid
Linear Collider Detector Description (LCDD)

- **GDML (Geometry Description Markup Language)**
  - Currently developed as part of LCG applications area
  - Tool available to generate GDML from any Geant4 program

- **LCDD**
  - Format developed for International Linear Collider (ILC) simulations
    - but generally applicable
  - Extends GDML to define full detector description

---

<table>
<thead>
<tr>
<th>Sensitive Detectors</th>
<th>LCDD</th>
<th>GDML</th>
</tr>
</thead>
<tbody>
<tr>
<td>Readouts</td>
<td>Expressions (CLHEP)</td>
<td></td>
</tr>
<tr>
<td>Regions</td>
<td>Materials</td>
<td></td>
</tr>
<tr>
<td>Physics Limits</td>
<td>Solids</td>
<td></td>
</tr>
<tr>
<td>Visualization Attributes</td>
<td>Volumes</td>
<td></td>
</tr>
<tr>
<td>Magnetic Fields</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---
org.lcsim: Compact Geometry Description

- LCDD can describe essentially any detector
  - But very verbose
  - Changing e.g. # of layers in calorimeter can result in many changes in LCDD file
- org.lcsim uses “Compact Geometry Description” to define detector
  - Simple XML format for describing ILC detectors
  - Handles typical ILC detector geometries
    - Range of detectors handled is extensible (by writing Java modules)
- Allows rapid prototyping of new detector geometries
- Does not require network access or installation of database software to run
- Automatic generation of full Geant4 LCDD geometry for full compatibility with SLIC
Detectors: Compact XML Example

Two layer stacks in an ECAL barrel

```xml
<detector id="2" name="EMBarrel" type="CylindricalBarrelCalorimeter"
  readout="EcalBarrHits">
  <dimensions inner_r = "150.1*cm" outer_z = "208.0*cm" />
  <layer repeat="20">
    <slice material = "Tungsten" thickness = "0.25*cm" />
    <slice material = "G10" thickness = "0.068*cm" />
    <slice material = "Silicon" thickness = "0.032*cm" sensitive = "yes" />
    <slice material = "Air" thickness = "0.025*cm" />
  </layer>
  <layer repeat="10">
    <slice material = "Tungsten" thickness = "0.50*cm" />
    <slice material = "G10" thickness = "0.068*cm" />
    <slice material = "Silicon" thickness = "0.032*cm" sensitive = "yes" />
    <slice material = "Air" thickness = "0.025*cm" />
  </layer>
</detector>
```
Small Java program for converting from compact description to a variety of other formats
org.lcsim Conditions Data

- Provide access to a extensible set of conditions for each detector including:
  - Detector Geometry
  - Algorithm Specific Constants
    - E.g. FastMC smearing parameters
- Doesn’t make assumptions about format of data
- Doesn’t rely on internet access, or local database installation
  - Downloaded and cached on first use
- Detector Constants stored in .zip file
  - Typically contains:
    - Compact geometry file
    - Set of (ascii) constants for standard algorithms
  - Can additionally contain:
    - Arbitrary files (xml, ascii, binary) needed by other algorithms
    - Other geometry formats (HepRep, LCDD)
    - Full fieldmap
- To define a new detector just create a new .zip file.
Available Detector Descriptions

- Although detector descriptions can live anywhere we maintain a CVS repository of detector descriptions
  - Exported to lcsim.org web site for automatic download
- 40 detector variants as of July 2006
- Many SiD variants, but also some GLD, LDC

- Anyone can contribute more
org.lcsim Goals

- “Second generation” ILC reconstruction/analysis framework
  - Builds on hep.lcd framework used since 1999
  - Full suite of reconstruction and analysis tools
- Uses LCIO for IO and as basis for simulation, raw data and reconstruction event formats
  - Isolate users from raw LCIO structures
  - Maintain full interoperability with other LCIO based packages
- Detector Independence
  - Make package independent of detector, geometry assumptions so can work with any detector
  - Read properties of detectors at runtime
- Written using Java (1.5)
  - High-performance but simple, easy to learn, OO language
  - Enables us last 10 years of software developments in the “real world”
- Ability to run standalone (command line or batch) or in JAS3 or IDE such as Netbeans, Eclipse
Why Java?

- Java is a pure Object Oriented Language
  - Simpler to learn and use than C++
    - Language design emphasizes ease-of-use over performance
    - Garbage collector takes care of freeing unused objects
      - Avoids distorting OO design by avoiding need for “ownership”
    - Very powerful standard library
      - Large number of open-source libraries including libraries for scientific computing
    - Platform independent, compile once just runs everywhere
      - Linux, Windows, Mac OSX)

- Physicist gets to concentrate on writing clean OO code to perform analysis tasks
  - Not understanding core dumps and learning difference between a pointer and a reference.

- Performance of Java code is close to that of C++
  - Dynamic (runtime) optimization can take into account actual usage patterns
    - not available to static optimizers used by Fortran, C++
  - Garbage collection often more efficient than user malloc/free (or new/delete)

- Java is mainstream language
  - Taught in university courses
  - Used by majority of sourceforge “open-source” projects
Why Java?

- Full access to runtime information makes interface to scripting languages easy
  - Jython, JRuby, Pnuts, …
- Open-source Java is rapidly becoming a reality
  - gcj (GNU), Harmony (Apache), Sun

Wide availability of Tools

- Several very powerful, free, IDE’s now available:
  - E.g. Netbeans, Eclipse, (IDEA)
  - Support editing, code completion, GUI building, debugging, performance profiling, refactoring, version control (CVS, Subversion), etc…
- Advanced tools such as maven (Apache) make project management easy
  - Maven is a Java based project management tool
  - After checking out code, single command “maven”
    - downloads dependencies,
      - Required libraries
      - Test Data
    - compiles code
    - runs test suite
    - deploys code
  - Maven can be integrated into IDE’s like Netbeans
Why Java? – Netbeans IDE
(After installing Java, cvs, maven)
- cvs –d :pserver:anonymous@cvs.freehep.org:/cvs/lcsim co GeomConverter
- cd GeomConverter
- maven
- cd ..
- cvs –d :pserver:anonymous@cvs.freehep.org:/cvs/lcsim co lcsim
- cd lcsim
- maven jar:install jas:install
Org.lcsim Reconstruction

- Reconstruction package includes:
  - Physics utilities:
    - Jet finders, event shape routines
    - Diagnostic event generator, stdhep reader/translator
    - Histogramming/Fitting/Plotting (AIDA based)
    - Event Display
    - Processor/Driver infrastructure
  - Fast MC
    - Directly reads stdhep events (or LCIO events)
    - Track/Cluster smearing
    - Produces ReconstructedParticles
  - Reconstruction
    - Cheaters (perfect reconstruction)
    - Detector Response
      - CCDSim, Digisim
    - Clustering Algorithms
      - Cheater, DirectedTree, NearestNeighbour, Cone
    - Tracking Finding/Fitting Algorithms
      - TRF, SLD Weight Matrix, Kalman filter
  - Muon Finding, Stepper
  - Vertex Finding (ZvTop)
**org.lcsim: Contrib Area**

- Goal of org.lcsim is not to provide “A single reconstruction package” but rather a framework into which reconstruction algorithms can be plugged.
- We encourage users to contribute code to the “contrib” area as soon as possible.
  - Important to encourage collaboration, reuse, and as learning tool.
- Contributions from: SLAC, Fermilab, Berkeley, NIU, Brown, Colorado, Colorado State, Santa Cruz, Iowa, Kansas, Kansas State, Oregon, Penn, Michigan, UT Arlington, …
  - Many contributions added in last year:
    - HMatrix cluster analysis
    - Vertex Fitter
    - Particle Flow (PFA) algorithms/template
    - SODTracker
    - Garfield Tracker
    - Calorimeter Cell Ganging
    - FastMC improvements
    - Tracking finding/fitting
    - MIP Finder
    - Minimum Spanning Tree Clustering
org.lcsim results

(See Norman Graf’s talk tomorrow)
org.lcsim Contributors

- SLAC, Fermilab, Berkeley, Santa Cruz, Iowa, Kansas, Kansas State, Oregon, Penn, Michigan, UT Arlington
Using org.lcsim with JAS3

- The org.lcsim can be used standalone, with an IDE, or inside JAS3. Same code can be used in all modes, so easy to move back and forth
  - E.g. develop in IDE and run in JAS3
  - E.g. develop in JAS3 and run in batch

- JAS3 org.lcsim plugin adds:
  - Example Analysis Code
  - org.lcim Event browser
  - Easy viewing of analysis plots
  - WIRED event display integration
org.lcsim: Examples

These examples are written using the Java language. After opening them you need to compile and load them, and then use their data to them using the Run menu.

org.lcsim examples

These examples are written using the Java language. After opening them you need to compile and load them, and then use their data to them using the Run menu.

org.lcsim Jython examples for advanced users

These examples are written in Jython. They have to be executed from within the frame of executing Java examples as well. You will have to provide data sample Tutorial visit Writing a Jython Driver.

mainLoop.py

The main Jython wrapper to load any other Java or Jython example.

Analyst101.py

A modified Jython version of Analyst101 java. Analyst101.java is simultaneously run.

Examples (12).java

import org.lcsim.util.aida.AIDA;
import java.util.List;
import java.util.Vector;
import org.lcsim.event.EventHeader;
import org.lcsim.event.MCParticle;
import org.lcsim.event.Driver;

public class Analyst101 extends Driver {
    private AIDA aida = AIDA.defaultInstance();
    public void process(EventHeader event) {
    } // Get the list of MCParticles from the event
    List<MCParticle> particles = event.getMCParticle.class, event.MC_PARTICLES);
    // Histogram the number of particles per event
    aida.cloud<U>(tracks).fill(particles.size());
    // Loop over the particles
    for (MCParticle particle: particles) {
        aida.cloud<1>(energy).fillParticle(energy);
        aida.cloud<2>(cosTheta).fill(Vec3D.cosTheta(particle.getMomemtum()));
        aida.cloud<3>(phi).fill(Vec3D.phi(particle.getMomemtum()));
    }
}
org.lcsim: Examples
org.lcsim: Plot Viewing
Using org.lcsim with WIRED4
Using org.lcsim with WIRED4
Using org.lcsim with WIRED4
Using org.lcsim with WIRED4
Using org.lcsim with WIRED4
Interoperability: Event Display

Z Higgs ($M_H=120$ GeV) → same simulator, three different full detector geometries
Using JAS3 and org.lcsim on the Grid

- In collaboration with Tech-X we have developed prototype “Interactive Parallel Analysis” system.
  - Allows interactive analysis of (ilc) data using a farm of machines to run analysis in parallel
  - Maintains full interactivity using JAS3
Sending Code and Merging Results

1. Download the dataset from the repository
2. Split the dataset into 'n' chunks
3. Place them in the Data Store

Analysis Studio (JAS3)

Internet

Resource Manager (GRAM)

Stage Data

Merged Results

Data Store

Stage Code

Interactive Dataset Analysis Service

Connect

Browse & Search

Dataset Catalog

Choose Dataset

Send Analysis Partial Results
Secure Login + Catalog Browser
Catalog Service to Browse and Search Datasets
Integrated Development Environment
Interactive Graphical Results in Seconds

Controls to Start/Stop Analysis

Can change analysis code, clear graphs, and restart
How hard is it to get started with org.lcsim?

- Works on Linux, MacOSX, Windows
  - Should take about 15 minutes to install JAS3 and org.lcsim plugin.
- Case Study: SLAC Summer student
  - 2 semesters of Java experience
    - (no C++, Fortran etc)
  - Using tutorial on lcsim.org Wiki; installed software, downloaded data, and got useful results in one day (and fixed a few errors in the documentation along the way).
  - Regular analysis updates have been appearing on her blog ever since!
- Even if you don’t have Java experience you can get started almost as fast
  - (the only thing you will miss is the core dumps)
- Start here:
  - https://confluence.slac.stanford.edu/display/ilc/lcsim+Getting+Started
  - Problems? Attend Tuesday afternoon “Simulation” phone meeting or use discussion forum at http://forum.linearcollider.org/
LCIO: Data Samples

- LCIO data samples available via anonymous FTP
- Data sets
  - ILC500
    - 500 GeV machine parameters
  - ILC1000
    - 1 TeV machine parameters
  - singleParticle
    - Single particle diagnostic events
  - Zpole
    - Zpole diagnostic events

- **[event type]** - complex or single particle event type, e.g. ZZ, ZPole, muons, etc.
  - **stdhep** - input StdHeP files used to generate the events
  - **[detector name]** - detector geometry tag, such as **sidaug05**
    - **[data file format]** - output datafile format, e.g. LCIO or SIO
    - **[simulator]** - simulator that generated the events, e.g. lced4, slic, lelaps, mokka, etc.
    - **logs** - simulator job logs
Becoming an org.lcsim developer

- To get started you just need “Java”, “cvs”, “maven”
  - Maven is a Java based project management tool
  - Single command “maven”
    - downloads dependencies, compiles code, runs tests, deploys code

- All code in CVS

- To check-out and build all code:
  - set CVSROOT="pserver:anonymous@cvs.freehep.org:/cvs/lcsim"
  - cvs co GeomConverter
  - cd GeomConverter
  - maven
  - cd ..
  - cvs co lcsim
  - cd lcsim
  - maven

- Find more documentation at:
  - [http://lcsim.org/](http://lcsim.org/)
  - Read/Contribute to the Wiki at: [https://confluence.slac.stanford.edu/display/ilc/Home](https://confluence.slac.stanford.edu/display/ilc/Home)
  - Discuss at: [http://forum.linearcollider.org/](http://forum.linearcollider.org/)

- We strongly encourage developers to use IDE
  - Netbeans, Eclipse both free, easy to learn, very powerful
  - Use mevenide to teach IDEs about maven system
  - Instructions for installing Netbeans in out Wiki (confluence).
Interoperability – Next Step

- LCIO has been very successful in providing some interoperability between disparate ILC tools
- Obvious next step is to attempt “common geometry” system
  - org.lcsim and Marlin (GEAR) geometry already very similar
  - Will have small workshop at SLAC next week to discuss directions
  - Follow up workshop at DESY at beginning of November
    - Goal (ambitious) too have something to discuss/show at ECFA workshop in Valencia in November
  - This might be a good area for someone interested in ILC software to get involved.
- Java -> C++ interoperability
  - Ability to call C++ (MarlinReco) modules from org.lcsim??
    - Some experimentation ongoing with using SWIG to build glue code
    - See somewhat related (pre-alpha) project G4Java
Links

- lcsim.org - http://www.lcsim.org
- Wiki - http://confluence.slac.stanford.edu/display/ilc/Home
- org.lcsim - http://www.lcsim.org/software/lcsim
- Software Index - http://www.lcsim.org/software
- Detectors - http://www.lcsim.org/detectors
- LCIO - http://lcio.desy.de
- SLIC - http://www.lcsim.org/software/slic
- LCDD - http://www.lcsim.org/software/lcdd
- JAS3 - http://jas.freehep.org/jas3
- AIDA - http://aida.freehep.org
- WIRED - http://wired.freehep.org
Conclusion

- SLIC + org.lcscim provide complete framework for ILC detector studies
  - Many reconstruction algorithms exist
    - Plenty of work still to do done
      - Adding/improving algorithms
      - Performing physics studies and detector benchmarking
    - All software is developed using an “open-source” model
      - All code available in CVS (even before it is complete)
  - Anyone is welcome to become involved
  - Interoperability with other linear collider software is provided by using LCIO
    - Work is ongoing to further improve interoperability