SLAC-PUB-15296

LCIO: A Persistency Framework and Event Data Model for HEP

Steve Aplin, Jan Engels, Frank Gaede, Norman A. Graf*, Tony Johnson, Jeremy McCormick

***Abstract*—LCIO is a persistency framework and event data model which was originally developed for the next linear collider physics and detector response simulation studies. Since then, the data model has been extended to also incorporate raw data formats to support testbeam and real experimental data as well as reconstructed object classes for use in physics analyses. LCIO defines a common abstract programming interface (API) and is designed to be lightweight and flexible without introducing additional dependencies on other software packages. Concrete implementations are provided in several programming languages, providing end users the flexibility of using multiple simulation, reconstruction and analysis frameworks. Persistence is provided by a simple binary format that supports data compression and random event access. LCIO is being used by the ILC and CLiC physics and detector communities to conduct performance benchmarking studies such as the recently completed CLiC CDR and the ILC Detailed Baseline Design (DBD) study to be completed in 2012. Detector studies for the Muon Collider are also being conducted using LCIO as the event data model and persistency. Multiple test-beam collaborations have used LCIO to store and process hundreds of millions of events, providing experience with real data. Recently the Heavy Photon Search collaboration also adopted LCIO as its event data model and offline persistency format. In this talk we present details of its use in these various applications, and discuss the successful cooperation and collaboration LCIO has enabled. We will also present the design and implementation of new features introduced in LCIO2.0.**

I. Introduction

LCIO is an event data model and persistency framework originally developed for simulation and test beam studies for the International Linear Collider (ILC). LCIO was developed to allow the exchange of data and algorithms among Linear Collider working groups and thus provide a basis for common software development.

A. Guiding Philosophy

The guiding philosophy behind the development of LCIO was to identify the key elements for an event data model appropriate to a colliding detector experiment. These would be the objects common to essentially every high-energy accelerator-based particle physics experiment. Having done that, it was important to target a simple persistency format, provide reference implementations in several languages, document it well enough to ensure future readability, and have no external dependencies. Having a common event data model and a common persistency format would allow the exchange of software and data, freeing up physicists to concentrate on the science and not the computing.

Manuscript received November 19, 2012. This work was supported in part by the U.S. Department of Energy.

N. A. Graf*, T. Johnson and Jeremy McCormick are with the SLAC National Accelerator Laboratory, Menlo Park, CA 94025 USA (* corresponding author, telephone: 650-926-5317, e-mail: Norman.Graf@slac.stanford.edu). S. Aplin, J. Engels and F. Gaede are with the DESY National Laboratory, Notkestraße 85, 22607 Hamburg, Germany.

B. Requirements

LCIO has to define a data model that fulfills the current needs of the global linear collider community for ongoing simulation and test beam studies. As Java, C++ and some legacy Fortran are used in ILC software, LCIO has to provide user interfaces in all three languages. It also has to provide a performant persistency format, one which is both compact and fast. In order to make it easy for existing frameworks to adopt LCIO it has to be lightweight and flexible without introducing additional dependencies on other software packages.

C. Implementation and Design

LCIO defines a common API for Java and C++ using the AID [3] tool. Two independent implementations exist for Java and C++ in order to benefit from either languages advantages. As is common practice today, user code is completely separated from the actual I/O format and code, so more advanced formats (such as hdf5 [4] or root [5]) can be incorporated in the future. In order to support legacy software a Fortran API to LCIO is provided through a set of wrapper functions to the C++ implementation. Details of the design and implementation are described in reference [1]. A schematic overview of the software architecture is shown in Figure 1.

D. I/O Format

SIO (Simple Input Output) has been chosen as a first I/O format for LCIO. It offers on-the-fly data compression and pointer retrieval. Random access to the events in the file is provided by an embedded catalog of pointers.

E. Users

LCIO has become the de-facto standard for ILC software development both within the detector concept groups for physics and detector response simulations as well as by the detector R&D collaborations such as the TPC and Calorimeter test beam groups.

Presented at IEEE 2012 Nuclear Science Symposium, Medical Imaging Conference
Anaheim, California, October 29 - November 3, 2012

Work supported by US Department of Energy contract DE-AC02-76SF00515.

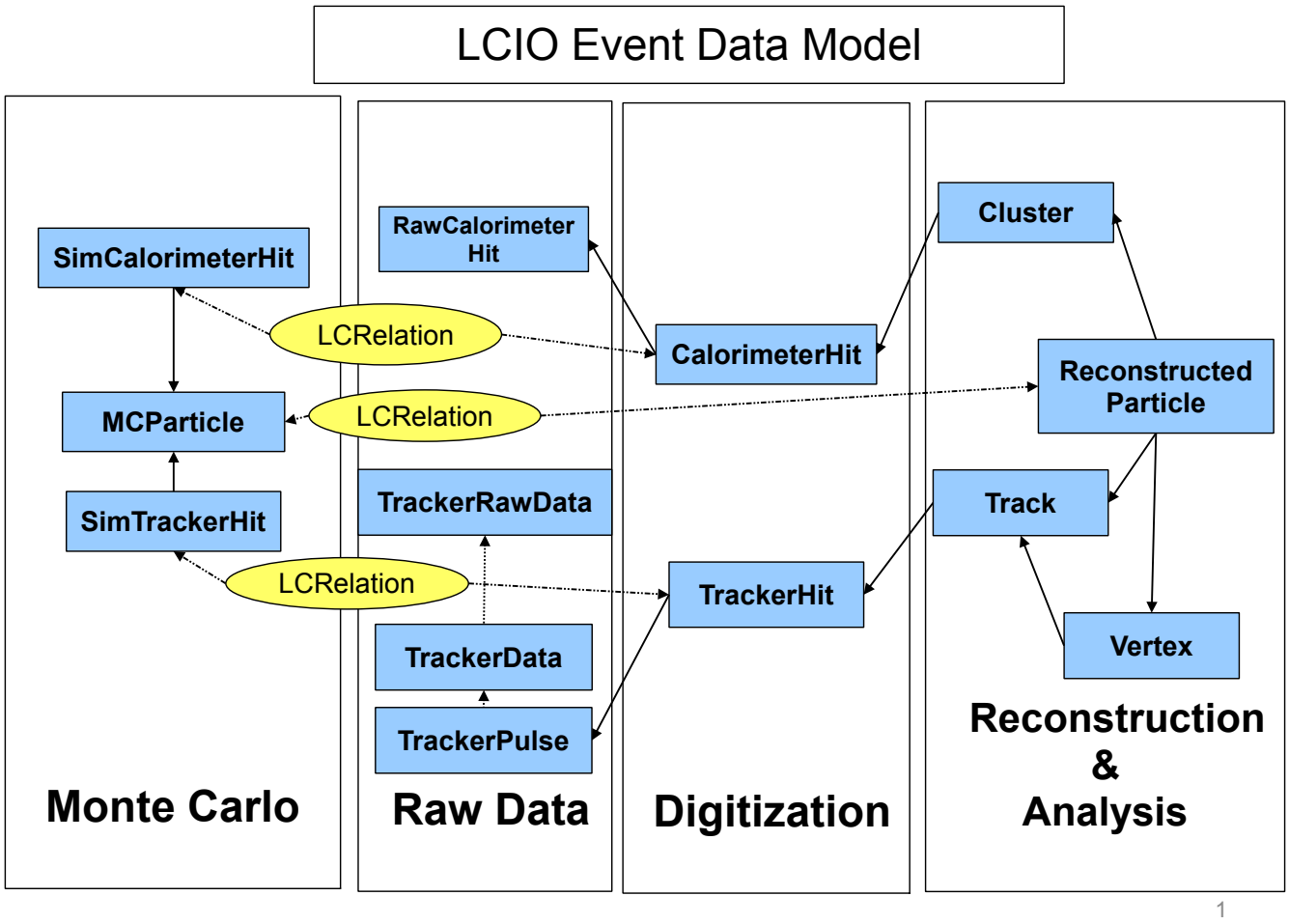

Fig. 1. The LCIO event data model showing the relationship between the main data classes. The full API can be found on the LCIO web site [2].

II. Event Data Model

Central to the event data model design is the class LCEvent that serves as a container for all data that are related to one event. It holds an arbitrary number of named collections (class LCCollection) of data objects (class LCObject). Run-related information is stored in LCRunHeader. Run-headers, events and collections have an instance of LCParameters that allows one to store arbitrary metadata. In figure 1 the major LCIO data entities are shown with the implemented relationships between the objects. In the following sections we describe the entities that are defined at every processing level from the Monte Carlo generator to analysis.

A. Monte Carlo

The main class at the Monte Carlo level is MCParticle. There will be exactly one collection with name MCParticle in every event that holds the Monte Carlo truth particles as created by the generator program. Particles that are created during simulation will be added to the existing list of MCParticles. By default, adding particles with their correct lineage ceases when a particle decays or interacts in a non-tracking region. Otherwise the number of MCParticles would explode in calorimeter shower development. Two generic hit classes, one for tracker and one for calorimeter hits are used to store the simulated detector response. All energy depositions are assigned to particles in the list, i.e. particles seen in the tracking region or that entered the calorimeter and started a shower. If a particle from a calorimeter shower is scattered back into the tracking region it is also added to the list with the resulting tracker hits assigned. A simulator status word is used to store the details about creation, interaction and decay of the particle. For the MCParticle, only parent relationships are stored. When reading the data back from the file the daughter relationships are reconstructed from the parents. This is to ensure consistency. Care has to be taken when analyzing the particle tree. Because a particle can have more than one parent the particle list in fact does not consist of a set of trees (one for each mother particle) but forms a directed acyclic graph. Thus the user has to avoid double counting in his code. Of course this only matters at the parton level as real particles do not have more than one parent.

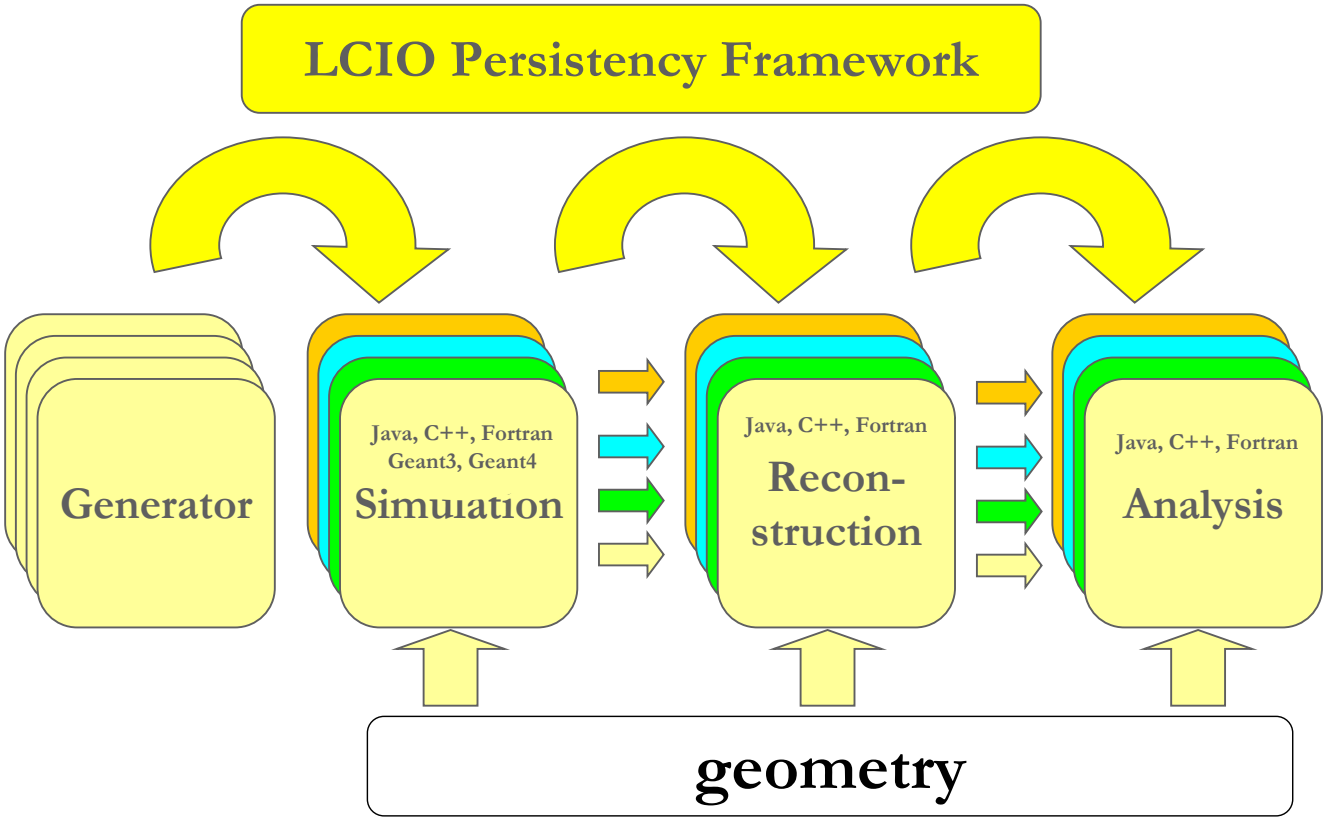

Fig. 2. Schematic showing how LCIO enables interaction between high-energy physics software written in different languages or frameworks.

B. Raw Data

The classes at the raw data level have been introduced to make LCIO also suitable for storing real data from test beam prototypes or running experiments. For calorimeters there is the class RawCalorimeterHit that consists of an ID, an integer amplitude and a time stamp. It is suitable for most calorimeter type detectors which integrate the energy deposition and do not record the full time evolution of the signal. This is different from the situation for the tracking subdetectors. There the data formats will vary considerably with the particular type of the tracking device, but in general more time information is recorded. Therefore the TrackerRawData class contains an array of integer measurements corresponding to the digitized time spectrum of energy deposits. If needed the raw data classes can also be used in simulation where one can link back to the Monte Carlo hits through special LCRelation objects.

C. Digitization

At the digitization level there are again two generic types of classes for tracker and calorimeter type subdetectors respectively. These classes will contain hit data after digitization and feature extraction. CalorimeterHit and TrackerHit are the types to be used in reconstruction and analysis code for Monte Carlo and test beam data. Support for one-dimensional (e.g. silicon microstrip or scintillating fiber detectors), two-dimensional (e.g. silicon pixel detectors) or full three-dimensional hits (e.g. TPC) is provided. If needed, users can access the corresponding original information for either data type.

D. Reconstruction and Analysis

Four main classes are defined at the reconstruction and analysis level. Hits are combined into Clusters and Tracks by pattern recognition and reconstruction algorithms. Both classes point back to the contributing hits. Clusters can also be combined from other clusters allowing a tree-like structure, e.g. one could build clusters with a geometrical algorithm and then combine some of these clusters to particles applying some track-match criterion. Due to the imaging capabilities of the planned Linear Collider calorimeters, clusters have an intrinsic direction assigned to them. Tracks are meant to represent the trajectories of charged particles and contain an ordered list of hits, along with an arbitrary number of TrackStates, which represent the best estimate for the track fit at those

TrackState locations. ReconstructedParticle is the class to be used for every object that is reconstructed. This can be a single particle like a track identified as a pion or a compound object like a jet made from many particles. ReconstructedParticle has lists of Tracks, Clusters and ReconstructedParticles that have been combined to form this particle. Besides the kinematics including the corresponding covariance matrix any number of hypotheses describing the particle's identity (PID) can be stored. ReconstructedParticle is intended to be basis for most physics analyses, where only rarely the need arises to go back to tracks, clusters or even hits.

E. Relation Objects

In order to be able to relate objects with each other that do not have a built-in relation in the LCIO data model the LCRelation is used. Typically LCRelation objects are used to store the links between raw data and the Monte Carlo truth information. This ensures that there is a clear separation between data classes that are to be used in analysis and reconstruction and the Monte Carlo classes that are used for developing and checking algorithms. LCRelation objects can also be used to relate reconstructed objects back to the Monte Carlo truth, e.g. if the hits are dropped from the files in order to save disk space. LCRelations can also be used to store transient links between objects at runtime. The relations are completely generic, and support one-to-one, one-to-many and many-to-one mappings in a way which is completely transparent to the end user.

F. User Extensions

The LCIO data model as described above has been designed in a way that it should fulfill all the current needs for ongoing Linear Collider studies. Care has been taken to make the data model flexible enough so that it can be used for a variety of different subdetector types. Nevertheless users will occasionally need to store information that is specific to a detailed aspect of their ongoing work and that is not foreseen in the data model. This can be done by using collections of LCIntVec, LCFloatVec and LCStringVec objects, i.e. arbitrary vectors of type int, float and string. Even though this mechanism is fairly generic it can be become somewhat cumbersome to handle user extensions, in particular if the information at hand involves more than one data type. For such cases, LCGenericObject allows users to store any self-defined class with LCIO that implements that interface. Every LCGenericObject has an arbitrary number of int, float and double attributes, where the numbers might be fixed among one collection or vary from object to object. Data stored in LCGenericObjects can be retrieved from an LCIO file either by using the user class implementation or through the generic interface. Thus to read an arbitrary LCIO file no additional knowledge or library is needed. This is different from other persistency systems, where typically a dictionary with the class definition is needed.

G. Transient Collections

The LCIO data model has been designed such that it can also serve as the transient data model in an application. Listener objects support a modular design of such applications, where every module gets an LCEvent with all the collections existing at that point and adds one or more collections with its result to the event. Typically anything added to the event will be made persistent. However some intermediate collections might only serve as input to a computation performed by a subsequent module. These LCCollections can be flagged as transient and will not show up in the output stream.

H. Default Collections

The LCEvent allows one to store an arbitrary number of named collections of the same data type. For example there can be several lists of Track objects available in one event. Typically there will be a collection of tracks or track segments for every tracking subdetector and one collection that holds all combined track fits. This collection is the one that will be used for most analyses. In order to make it simple for the end user, this collection can be flagged as being the default list for Tracks. In general there should be exactly one default collection for every type. This list should be complete in the sense that all known information has been taken into account and unique in the sense that it does not double count energy.

I. Metadata

LCIO is being used by a number of groups to store data for detectors with different features and capabilities. Thus it is necessary to describe the data that is stored in LCIO in a way that it can be interpreted by users from other groups without additional documentation and ideally even without modifying existing code. This metadata description can now be stored as arbitrary named parameters of types int, float and string attached to run-headers, events or collections. A number of predefined attribute names exist that are used to describe and interpret type information in LCObjects, such as the type of ReconstructedParticles or Clusters. These attributes can be parsed and interpreted by applications whereas other - user defined - attributes can at least be printed and interpreted by another user, making additional sources of documentation unnecessary.

J. Use with root

Many high-energy physicists have become accustomed to using root [5] for their analysis needs. Although some effort has been devoted to providing an LCIO persistency binding using native root classes, the resulting files were found to be larger and less performant than those based on SIO. Therefore, the recommended way to access LCIO files via root is to use the class dictionary which is provided as part of the LCIO release.

III. LCIO AND CODE RE-USE

The Linear Collider user community is broad and diverse, and it should come as no surprise that multiple software frameworks have been developed to support the simulation, reconstruction and analysis efforts. Normally, such a disjoint

architecture would preclude close cooperation and collaboration. But having both a common event data model and a common persistency format has allowed the community to use the best tools from each toolkit without consideration of language or platform. Figure 2 illustrates how LCIO provides a backbone for various applications. For instance, events can be generated in FORTRAN, simulated in C++, reconstructed in Java, and analyzed with root, all because of the common event data model and persistency.

IV. Other Users

A. HPS

The Heavy Photon Search (HPS) [6] is a fixed-target experiment at Jefferson Lab aimed at discovering a hidden-sector, heavy photon. Electron-positron pairs produced in the decay of such a particle are momentum-analyzed using silicon strip detectors inside a dipole magnetic field and their energy is measured in a crystal calorimeter array. The time to design and optimize the detector via simulations was extremely short, yet the need to precisely model the very high backgrounds very close to the beam was critical to the success of the recently completed test run. Although the detector is a fixed target configuration, the event data model still maps very well onto the detector elements and reconstruction needs. By adopting LCIO as the event data model and persistency format, the experiment was able to immediately benefit from the large amount of simulation and reconstruction software developed by the Linear Collider community.

B. Whizard Monte Carlo Event Generator

Whizard [7] is a Monte Carlo event generator for the LHC, ILC, CLIC, and other high-energy physics experiments. The events can be written to file in standard formats, including ASCII, STDHEP, the Les Houches event format (LHEF) or HepMC. However, although there is a standard persistency format defined for the STDHEP files, the LHEF and HepMC standards define only the in-memory API. It is left up to individual experiments to define their own persistency format, which severely limits the exchange of generated events, or forces the adoption of unwanted external dependencies. The developers are therefore considering LCIO as the default persistency format.

C. Data Preservation and Archiving

The field of high-energy physics does a very poor job of preserving its data beyond the publications which arise out of analyses internal to individual collaborations. One of the stumbling blocks is the custom persistency format (often undocumented beyond the code which actually writes the data), and the difficulty maintaining very large code bases to read the data and access the information. We believe that the simplicity and completeness of the design and implementation of LCIO makes it a good candidate for a common solution to common archival requirements in the accelerator-based high-energy physics community.

V. Summary and Outlook

Since its first public release in November 2003 LCIO has been adopted by a number of groups and has become a de facto standard for Linear Collider software. The current release provides the complete data model including reconstruction objects and user extensions. Having a common event data model has enabled an unprecedented level of cooperation and collaboration across disparate, and sometimes competing detector concepts, across languages (Fortran, Java, C++, python), across platforms (Linux, Mac, Windows) and across regions (America, Europe, Asia). The use of well-defined interfaces for data exchange has been more important than imposing a single framework, language or platform. As such, LCIO has been very successful in the Linear Collider community, both for Monte Carlo simulations and real data applications. It is beginning to attract users in other communities (such as HPS, the Whizard Monte Carlo program) and would be an excellent candidate for data preservation and archival projects.

References

[1] F.Gaede, T.Behnke, N. Graf, T. Johnson CHEP03 March24- 28, 2003 La Jolla, USA Conference proceedings, TUKT001, arXiv:physics/0306114.
[2] LCIO Homepage: http://lcio.desy.de/
[3] AID Homepage: http://java.freehep.org/aid/index.html
[4] http://www.hdfgroup.org/HDF5/
[5] http://root.cern.ch
[6] https://confluence.slac.stanford.edu/display/hpsg/Heavy+Photon+Search+Experiment
[7] http://whizard.hepforge.org/