Design considerations for the LCIO framework.

Frank Gaede, DESY -IT-

January 22, 2003

Abstract

This document summarizes some of the requirements for the LCIO persistency framework and their implications on the design of the interfaces. A schematic proposal for the design is presented.

1 Introduction

The LCIO library will be realized in Java and C++ with an additional Fortran interface based on the C++ version. The common user interface will be defined using the AID (Abstract Object Definition) Tool generating Java and C++ skeleton files. This insures the two user interfaces are going to be in sync at all times. Some basic design decisions have to made, concerning the package/compilation unit structure and the set of common interfaces provided to the users for the different tasks at hand. These are described in the next chapter.

2 Use cases

There are some obvious basic uses cases for a persistency framework for simulation studies:

2.1 Existing (simulation) applications

Existing simulation applications that want to use LCIO as an (additional) output format will already have a set of event data classes (hit collections etc.) defined. There needs to be a straight forward way to use LCIO by implementing some *small* interface for all classes that should be made persistent. No read access to LCIO is needed.

2.2 Applications that extend the data (reconstruction)

Applications that want to extend some existing data set, e.g. a reconstruction program have additional needs. They have to be able to read the data and add the reconstruction result to the event. The original data should not be modifiable (at least not using the provided API, one can allways mess with pointers in C++). For the data added to the event they might want to use existing classes (by implementing a small interface) or use predefined classes from LCIO.

2.3 Read only applications (analyses)

The simplest case is a read-only application like an analysis program (assuming the analysis output will be in some different format (AIDA, JAS, root,...). There should be a simple interface, that provides some convenience methods for the analysis, e.g. the transformation of parameters or the computation of derivable quantities. Modification of the data is prohibited.

2.4 New applications that want to use LCIO

New applications that want to use LCIO should be able to use default implementations provided by LCIO.

3 Requirements

Together with the use cases above on has the following list of requirements that have impact on basic design decisions:

- Need for an abstract object layer - user code will have no dependance on concrete implementation).
- Common implementation in Java and C++ as far as the user interface is concerned.
- Realization as a libray, i.e. the user should not make any changes to the classes in LCIO.
- Simple interface to write existing classes.
- Interface used for the read access to data, where additonal information might be added.
- Read only access for data analysis.

- Default implementations for new applications.
- Clearly structured packages. Users should just have to look only at the part of the API they are concerned with.
- Keep it simple - for the developers and the users.

4 Design proposal

This section proposes a design that fullfills the requirements mentioned above. The following packages (fig.1) structure the classes (fig.2) according to their dependencies and use cases:

- `hep.lcio.base`
  Base interfaces for writing objects. These have to be implemented by existing classes that are to made persistent. Either directly or using a decorator pattern in case users don't want to change their classes. There are only get-methods in the interfaces.
- `hep.lcio.event`
  Generic interfaces for all user code that involves reading - there are get-methods and methods for adding information to the event.
- `hep.lcio.analysis`
  Decorater classes that add functionality to the event data objects needed for analyses, so called convenient methods.
- `hep.lcio.impl`
  Default implementations of the interfaces in `hep.lcio.event`. These classes have set-methods as well and thus can be used for writing data to the LCIO store. They are going to be used for reading as well, but not made public to the user - so they can't modify existing event data.
- `hep.lcio.ioimpl`
  Subclasses that add IO mechanisms to the classes in `hep.lcio.impl`, e.g. for C++ and SIO friend declarations are needed. Classes in this package will be different for Java and C++, so there will be no common AID definition of their API and the API will not be public.
- `hep.lcio.io`
  Interface for the io-infrastructure in particular Reader and Writer classes and the extesnion mechanism.
- `hep.lcio.sio`
  Concrete implementation of `hep.lcio.io` for SIO. Will not be made public to the users. And Probably there will be different APIs for Java and C++.

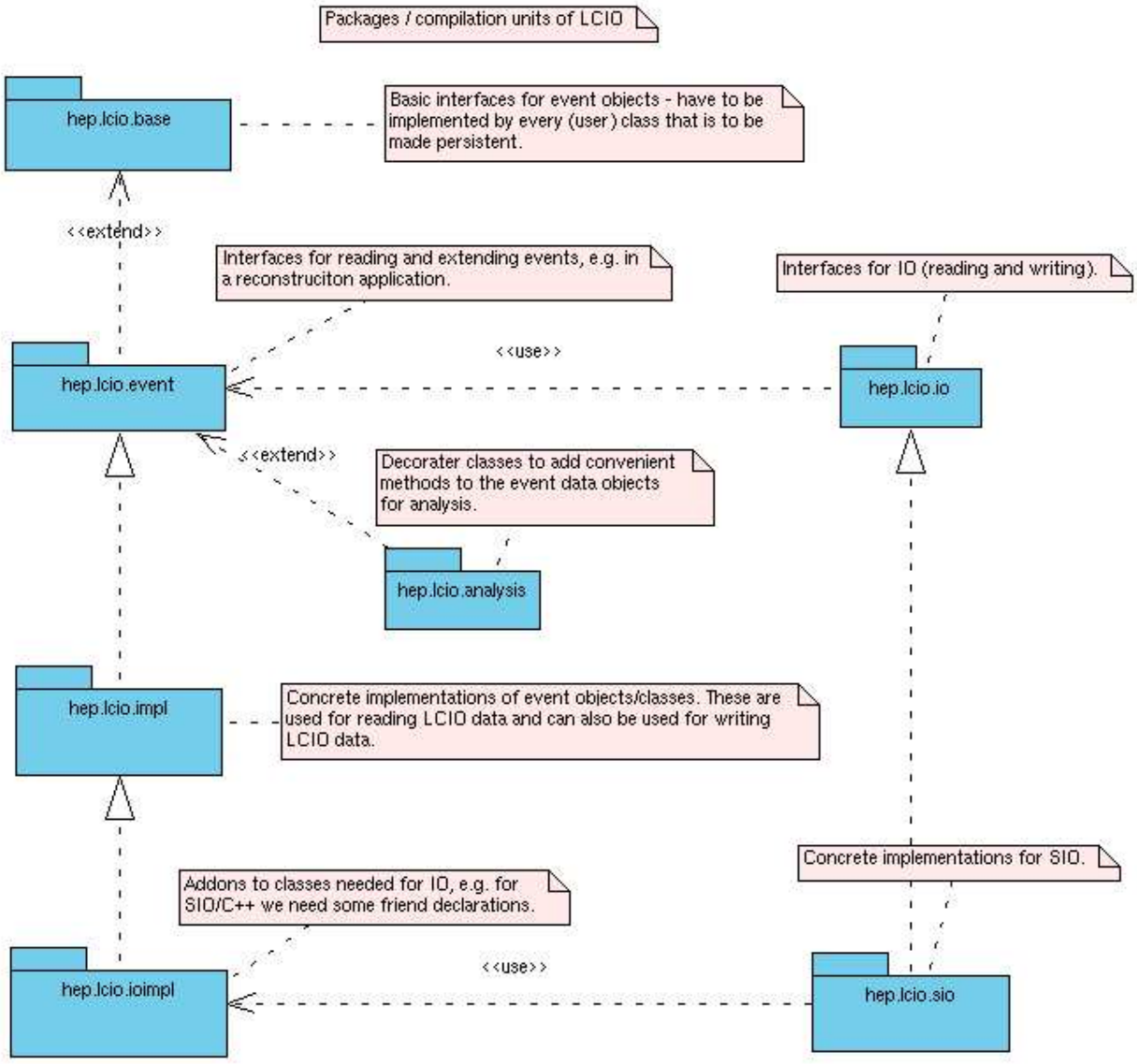

Figure 1: Overview of packages of LCIO.

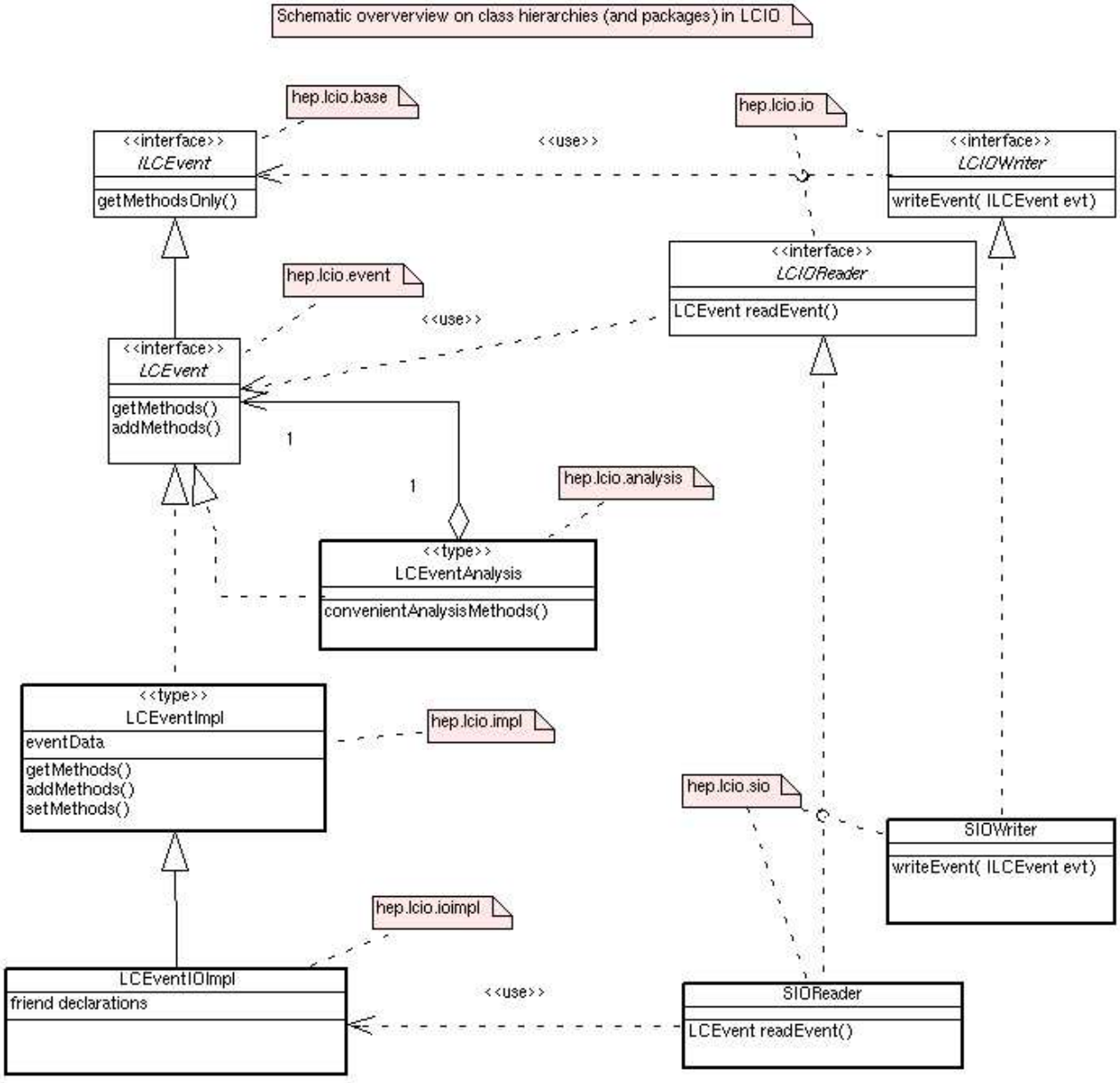

Figure 2: Schematic overview of classes in LCIO.