RAIDA:
ROOT Implementation of the AIDA Interface

Thomas Krämer

DESY Hamburg

ILC Physics and Software Meeting

Cambridge, April 5th, 2006

Outline

- Motivation and Introduction
- The AIDA Interface
- Available AIDA Objects in RAIDA
- Using RAIDA with Marlin
- Summary and Outlook

Introduction and Motivation

- reconstruction and analysis $\Rightarrow$ use the Marlin framework
  - read event and run data from LCIO file
  - process data (find tracks and clusters, do pflow ...)
  - write events to LCIO file or run event display
- developers and users need additional kinds of output from Marlin:
  - histograms to develop and debug reconstruction algorithms
  - plots to check performance of reconstruction after software changes: Marlin, Mokka, GEANT, GEAR, geometry, etc.
  - plots to control quality during Monte Carlo production
  - histograms and n-tuples for physics analyses
- replace the output processor to write any desired format (only useful for data stored in LCIO objects)
- histograms and n-tuples can be filled at any point of a Marlin processor
- dedicated check plots can be made in `check()` method of any processor

Which Tool to choose?

- traditionally a library is used to produce histograms or n-tuples

  $\Rightarrow$ results in dependency on specific library (implementation)

  $\Rightarrow$ switching libraries means changing user code

- produces tendency towards the usage of fancy features

Alternative:

- usage of the histogram and n-tuple functionality through abstract interface
  - code of application $\Rightarrow$ only depending on interface
  - using different library $\Rightarrow$ no changes in code
  - free choice of preferred tool or usage of two tools in parallel
  - choice of library while linking or via switch in program (at runtime)

What is AIDA?

Abstract **I**nterfaces for **D**ata **A**nalysis

- standard set of interfaces for creating and manipulating common physics analysis objects (histograms, n-tuples, fitters, IO etc.)
- goal: interfaces can be used regardless of analysis tool
  - users only need to learn one set of interfaces (even if they use more than one tool)
  - no changes in user code
- currently C++ and Java versions of the AIDA interfaces exist (they are as identical as the underlying languages permit)
- developers are themselves working on high-energy physics data analysis tools
- more information: `http://aida.freehep.org`

AIDA Implementations and Tools

- C++, Python, and Java implementations are available
  - AIDA-JNI: allows any Java implementation of AIDA to be used out of a C++ program, compatible with AIDA 3.2.1
    (more info: `http://java.freehep.org/aidajni`)
  - JAIDA: a Java stand-alone implementation of AIDA 3.2.1, part of the FreeHEP Java library (more info: `http://java.freehep.org/jaida`)
  - PAIDA: a pure Python stand-alone implementation of AIDA 3.2.1
    (more info: `http://paida.sourceforge.net`)
- analysis tool systems implementing the AIDA interfaces
  - PI: a project in the Application Area (AA) of the LHC Computing Grid (LCG) project supports AIDA 3.2.1 (more info: `http://cern.ch/pi`)
  - JAS3: a graphical data analysis toolkit based on AIDA
    (more info: `http://jas.freehep.org/jas3`)
  - Open Scientist: an architecture to offer an open, modular, free, portable, evolutionary, efficient and collaborative environment for doing data analysis, supports AIDA 3.2.1
    (more info: `http://www.lal.in2p3.fr/OpenScientist`)

Overview of AIDA Classes

- factories: instantiate new AIDA objects
- histograms: 1D, 2D and 3D binned histograms
- profile histograms: 1D and 2D binned profile histograms
- clouds: 1D, 2D, and 3D unbinned histograms, useful for scatter plots, rebinnable histograms and for unbinned fits
- tuples: arbitrary dimension n-tuples
- trees: arranging objects into folders, and for IO
- plotting: displaying plots
- functions: plotting functions and fitting
- fitter: perform binned and unbinned fits to the AIDA data storage objects

How to use AIDA?

- factories are used instead of creating objects with “new”
- a function is used to get a pointer to the desired implementation
- one “master” factory: `IAnalysisFactory`

  $\Rightarrow$ obtain from this all other factories

  - `ITreeFactory`: creates AIDA-trees
  - `IHistogramFactory`: histograms, clouds and profile histograms
  - `ITupleFactory`: n-tuples
  - ...

How to use AIDA?

sample code using a specific AIDA implementation

```
#include <AIDA/AIDA.h>

IAnalysisFactory *myAIDA = AIDA_createAnalysisFactory();
ITreeFactory *myTreeFactory = myAIDA->createTreeFactory();
ITree *myTree = myTreeFactory->create("outputFile.aida");

IHistogramFactory * myHistoFactory
   = myAIDA->createHistogramFactory(*myTree);
ITupleFactory * myTupleFactory
   = myAIDA->createTupleFactory(*myTree);

IHistogram1D *Histo
   = myHistoFactory->createHistogram1D("name","title",50,0,1);
ICloud1D *myCloud
   = myHistoFactory->createCloud1D("name","title",400);
   ...
```

From AIDA to RAIDA

- want for using ROOT to look at histograms, analyse n-tuples etc.
  $\Rightarrow$ Marlin has to produce output in ROOT format

Solution:

- implementation of AIDA using ROOT
- first approach is restricted to writing out objects
- **problem:**
  - some AIDA objects have more features than the corresponding ROOT objects
    (histograms: bin mean, bin entries)
  - objects do not exist within ROOT
    (ICloud, IDataPointSet, IAnnotation)
- use several ROOT objects to hold information usually not kept by ROOT
  (for 1D histograms save 3 TH1D to hold bin hight, bin mean and bin entries)
- substitute missing objects by other objects if possible
  (IClouds are saved as TTree if they are not converted to histograms)
- no implementation of objects which can not be “emulated”

Available AIDA Objects in RAIDA

- file/directory: `ITree` (memory and disk)
  `storeName(), cd(), pwd(), mkdir(), commit(), close()`
- histograms

`IHistogram1D`	`IHistogram2D`	`IHistogram3D`
`ICloud1D`	`ICloud2D`	`ICloud3D`
`IProfile1D`	`IProfile2D`	`IAxis`

  projections and slices will work soon
- n-tuples: `ITuple`
  no logical chain of `ITuples`, no `IFilter`, no `IEvaluator`, no projection
- there is no meaningful way of implementing `IDataPointSet` without writing an own ROOT class
- no equivalent construct for `IAnnotation` in ROOT

Building RAIDA

- download the software
  - visit our software portal: `http://ilcsoft.desy.de`
  - CVS repository via anonymous checkout

    export CVS_RSH=ccvssh

    export CVSROOT=:ext:anonymous@cvssrv.ifh.de:/ilctools

    ccvssh login

    cvs co -r v00-01 RAIDA

- set environment variables that describe your root installation

  export ROOTSYS=/opt/products/root/5.08.00

  export LD_LIBRARY_PATH=$ROOTSYS/lib:$LD_LIBRARY_PATH

  export PATH=$ROOTSYS/bin:$PATH

- build library with `gmake`

  this will create `./lib/libRAIDA.a`

Using RAIDA

- set proper environment for RAIDA

  export RAIDA_HOME=$home/ilcsoft/RAIDA/v00-01

  source $RAIDA_HOME/bin/aida-setup.sh

- to build your program with RAIDA you need the proper includes and libraries
  - return include path: `aida-config --include`
  - return library path: `aida-config --lib`

  (these scripts are available for all AIDA implementations)

- link also against ROOT: `root-config --libs`
- build your program
- when wanting to use RAIDA together with Marlin
  - Marlin v00-09-04 is adapted to use RAIDA v00-01
  - enable the AIDAProcessor in the steering file

    opens AIDA file, creates a directory for each processor to sort objects

Summary

- first ROOT implementation of the AIDA interface available: RAIDA
- combined usage with Marlin to produce check plots or n-tuples
- writes ROOT files with standard ROOT objects

Outlook

- things to be added
  - projection and slice functionality for histograms
  - implementation of functions which are needed for Marlin and Co.
  - inclusion of installation guide in the MarlinReco manual
- please use our software portal for feedback and bug reports:

  `http://ilcsoft.desy.de`