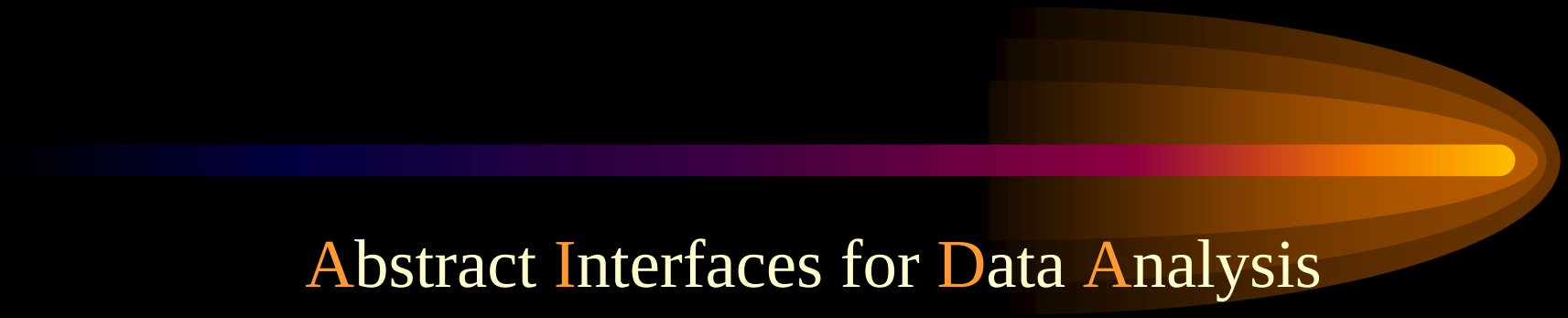
AIDA



Abstract Interfaces for Data Analysis

http://aida.freehep.org

C'est quoi ?

- Un API
- Une architecture
- Une philosophie

- Ce que le CERN a ou va rater.
- Ce que ROOT aurait du être

Historique / HEPVis'99 (LAL)

- Présents ; développeurs de Anaphe, Jas/FreeHEP, Hippodraw, OpenScientist.
- ROOT pas présent.
- On a déjà des implémentations (de package d'histogramme, plotting, etc…) couvrant plusieurs langages : C++, java.
- Certains veulent grouper les forces.

Historique / Comment faire ?

- Sachant que chacun ne veut pas dépendre des autres.
- OpenScientist : y en a marre du CERN (dictature de softwares que l'on doit débugger et que l'on ne peut pas faire évoluer).
- ANAPHE : « not done at CERN » syndrome (hélas).
- Jas/FreeHEP : pas de C++ !
- Hippodraw : depuis longtemps sur le problème (y en a marre du CERN aussi).
- ROOT : TROOT.h : « The ROOT of Everything » : tout est dit.

(Tout ce petit monde étant très « selfish oriented »).

Une solution élégante…

- Converger vers un lot d'interfaces abstraits.
- Proposé par Anaphe (A.Pfeiffer). Probablement inspiré de ce qui a été fait par P.Binko dans Gaudi / LHCb (architecture où les gens ont « vu » le rôle potentiel des interfaces abstraits).

  Le AIDA::IHistogram1D vient de Gaudi.
- Ils ont dit « ok » : OpenScientist, ANAPHE, FreeHEP / jas.

  Naissance de AIDA.
- Ils n'ont pas suivis :
  - hippodraw (hélas),
  - ROOT (pas là, mais n'auraient pas suivis de toute façon puisque ils n'ont pas « vu » l'intérêt et que l'idée était supportée par les « ennemis » du moment (Anaphe/ CERN-IT)).

Interface purement abstrait

- C’est quoi ?
- Une classe où toutes les méthodes sont virtuelles (pas de code).

```
class IHistogram1D {
    virtual void fill(double x,double w) = 0;
    virtual double binHeight(int index) = 0;
    ….
 }
```

Intérêt technique

- Intérêt technique fondamental dans les grosses architectures : permet de minimiser les couplages entres catégories.
- Exemple standard : Histogram / Plotter :

```
Class Histogram : IHistogram {
  virtual void fill (double x, double w) {…}
  virtual double binHeight(int index) {…}
}

libHistogram
```

```
#include <IHistogram.h>
Class Plotter {
  void plot(IHistogram& histo) {
    double w = histo.binHeight(i);
      …
  }
}

libPlotter
```

- libPlotter n'a pas à être linkée a libHistogram !!!!! Fort découplage.

Intérêt sociologique

- Un interface abstrait est le minimum de code sur lequel les gens doivent converger pour atteindre l'interopérabilité
- Donc c'est le vecteur idéal pour les gens qui veulent partager quelque chose mais ne sont d'accord sur rien.
- Permet aussi de fixer, dans du code, un API pour les physiciens sans pour cela dépendre fortement d'une implémentation, d'un langage, d'une technologie.
- Permet de fixer, dans du code, les users requirements et le « savoir faire d'un métier »
- Par exemple. C'est quoi un HEP histogramme ? Il suffit de regarder AIDA::IHistogram1D.h pour avoir une idée ; avec l'avantage que ce n'est pas du texte, mais du code sur lequel quelqu'un peut démarrer une implémentation.

AIDA

- Une cinquantaine d'interfaces qui peuvent être rangés en trois catégories :
  - data : histogram, tuple, function, cloud, data point set.
  - management : arborescence de conteneurs « à la file system » (le AIDA::ITree), les factories.
  - facilités : plotting, fitting et stockage (à travers le ITree).
- Les interfaces sont décrites dans des fichiers .aid. Une moulinette produit les interfaces pour : C++, Java, python
- Il existe une suite de testes et un tutorial fait par le team AIDA / SLAC.

Exemple C++

```
#include <AIDA.h>
int main () {
  AIDA::IAnalysisFactory* aida = AIDA_createAnalysisFactory();
  AIDA::ITreeFactory* tf = aida->createTreeFactory();
  AIDA::Itree* tree = tf->create('file.xml','xml',false,true);
  AIDA::IHistogramFactory* hf = aida->createHistogramFactory(*tree);
  AIDA::IHistogram1D* h = hf->createHistogram1D('h','title',100,0,1);
  while() {h->fill();}
  tree->commit();
  delete aida;
  return 0;
}
```

UNIX> c++ aida_exa.cxx `aida-config --cflags` `aida-config --libs`

AIDA_createAnalysisFactory est le seul point d'entrée concret !!! Elégant

Implémentations

- OpenScientist / Lab : C++. Open source + LAL.
- Jas / FreeHEP : java, SLAC.
- PAIDA : python (Japon). http://paida.sourceforge.net

- Anaphe : C++, CERN/IT transformé en LCG/PI.
- Mais LCG/PI veut en fait faire autre chose… (CERN mess as usual)

Clients

- Geant4 supporte les interfaces AIDA dans ses exemples. Les utilisateurs « directs » sont là.
- Gaudi utilise une partie : AIDA::IHistogram. Mais il y a un clash autour du management : le AIDA::ITree est en conflit avec le Gaudi « transient store ».
- E.Aubourg à Hourtin (java) « c'est maintenant utilisable »

ROOT & AIDA

- R.Brun : voir CHEP'03 : perspective on future data analysis in HENP :
  « In AIDA I don't believe »
- Il est intéressant de voir que dans ROOT il n'y pas d'interfaces purement abstraits. Pas sûr que Vador ait « vu » l'intérêt technique des interfaces. L'utilisation de la moindre classe virtuelle (TVirtualXxx) de ROOT (par exa TVirtualPad ou même TObject) oblige à s'attacher à 200 000 lignes de code (contenu de libCore + CINT) : not appealing.
- ROOT y gagnerait beaucoup en utilisant cette technique….
- Quant a la vision « sociologique » d'ouvrir l'engineering de ROOT…

- LCG / PI implemente AIDA sur ROOT.
- Jas / OpenScientist savent lirent les fichiers .root (sans ROOT) contenant des THs et TTuples. (OpenScientist sait aussi les écrire).

LCG & AIDA

- LCG / PI.
- Implementation sur ROOT : discutable.

```
namespace AIDA_ROOT {
   class Histogram : TH1, AIDA::IHistogram1D {
   };
 }
```

- User semantic layer. Couche non-abstraite au dessus de AIDA :

```
int main() {
   pi_aida::Histogram h('h','title',100,0,100);
   while() h.fill(i);
   ….
}
```

- Pourquoi pas. Mais aujourd'hui leur implémentation n'est pas sur les interfaces ! A discuter…
- Future de LCG / PI : Vador est maintenant le chef de LCG / AA….

Problèmes, joblist

- « extraire un interface » est un processus lent. C'est un problème technique mais aussi sociologique. Il a fallu plus d'un an pour avoir le jeu des classes pour l'histogramming. On aura probablement seulement maintenant des bons interfaces pour le tuple. (En utilisant les templates).
- Plotting. Travail en cours pour gérer les « styles » (il faut être exhaustif).
- Le AIDA::ITree est un problème pour les gens ayant déjà un data manager.
- AIDA::IStore ?
- User semantic layer ? Ok. Mais alors le groupe se dirige vers une bibliothèque (avec les problèmes associés : makefile, etc…). Cela doit être bien fait (en particulier couvrir java et python). Le code LCG / PI peut être un point de départ.

- AIDA::Dev::IXxx : interfaces pour l'ingénieur (pour vraiment régler le problème de l'interopérabilité).

Conclusions I

- L’utilisation d’interfaces purement abstraits est une clef importante.
- AIDA couvre le problème de l’analyse de données.
- Il faudrait faire la même chose pour le reste.
- géométrie : VGM (I.Hrivnacova, IPN)
- simulation :
  - Geant4 = ROOT : Il y a des G4VXxx (=TVirtualXxx) mais pas de Geant4::IXxx. Ils ne veulent pas le faire. Dommage, Geant4 y gagnerait en clarté dans son architecture.
  - VMC est un API commun vers plusieurs simulateurs, pas des interfaces abstraits.
  - ISIM : c’est à faire.

Conclusions II

- L'« extraction » des interfaces aurait du être la pierre et la couche fondatrice du soft LCG. Le package « BVB » Bosonic Virtual Bus.
- Mais hélas le CERN ne semble pas avoir pris la bonne direction
- Est ce que le reste du monde saura s'organiser pour sortir (enfin) de l'âge de pierre en matière de software ?
- Max Planck (à propos de la mécanique quantique) « le seul moyen pour que les idées nouvelles s'imposent est d'attendre que la génération précédente soit dans la tombe (bon disons à la retraite) » Pour nous, il faudra attendre 2015-2020 !