How LC software could profit from LHC software

TILC09, 17-21 April 2009, Tsukuba

P. Mato /CERN

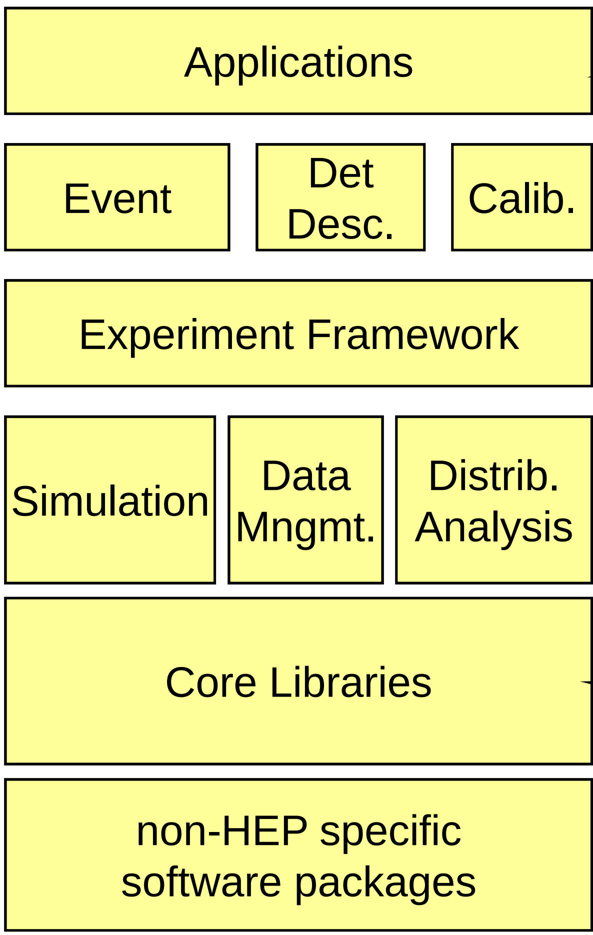
About myself

- Former LHCb core software coordination
  - Architect of the GAUDI framework
- Applications Area manager of the Worldwide LHC Computing Grid (WLCG)
  - Develops and maintains that part of the physics applications software and associated infrastructure that is common among the LHC experiments
- Leader of the Software Development for Experiments (SFT) group in the Physics (PH) Department at CERN
  - Projects: Geant4, ROOT, GENSER, SPI, CernVM, etc.

Introduction

- The main theme of this short presentation is a review what can be learn and reuse from the LHC software for the ILC
- Emphasis on interoperability, reuse and enabling independent developments
  - Level 1– Application Interfaces/Formats
  - Level 2 – Software Integrating Elements
- Review some of the interfaces and common packages developed in the context of LCG

Simplified Software Structure



Applications are built on top of frameworks and implementing the required algorithms (e.g. simulation, reconstruction, analysis, trigger)

Every experiment has a framework for basic services and various specialized frameworks: event model, detector description, visualization, persistency, interactivity, simulation, calibrarion

Specialized domains that are common among the experiments (e.g. Geant4, COOL, Generators, etc.)

Core libraries and services that are widely used and provide basic functionality (e.g. ROOT, CLHEP, HepMC,…)

Many non-HEP libraries widely used (e.g. Xerces, GSL, Boost, etc.)

Software Components

One or more implementations of each component exists for LHC

- Foundation Libraries
  - Basic types
  - Utility libraries
  - System isolation libraries
- Mathematical Libraries
  - Special functions
  - Minimization, Random Numbers
- Data Organization
  - Event Data
  - Event Metadata (Event collections)
  - Detector Description
  - Detector Conditions Data
- Data Management Tools
  - Object Persistency
  - Data Distribution and Replication
- Simulation Toolkits
  - Event generators
  - Detector simulation
- Statistical Analysis Tools
  - Histograms, N-tuples
  - Fitting
- Interactivity and User Interfaces
  - GUI
  - Scripting
  - Interactive analysis
- Data Visualization and Graphics
  - Event and Geometry displays
- Distributed Applications
  - Parallel processing
  - Grid computing

Programming Languages

- C++ used almost exclusively by all LHC Experiments
  - LHC experiments with an initial FORTRAN code base have completed the migration to C++ long time ago
- Large common software projects in C++ are in production for many years
  - ROOT, Geant4, …
- FORTRAN still in use mainly by the MC generators
  - Large developments efforts are being put for the migration to C++ (Pythia8, Herwig++, Sherpa,…)
- Java is almost inexistent
  - Exception is the ATLAS event display ATLANTIS

Scripting Languages

- Scripting has been an essential component in the HEP analysis software for the last decades
  - PAW macros (kumac) in the FORTRAN era
  - C++ interpreter (CINT) in the C++ era
  - Python is widely used by 3 out of 4 LHC experiments
- Most of the statistical data analysis and final presentation is done with scripts
  - Interactive analysis
  - Rapid prototyping to test new ideas
- Scripts are also used to "configure" complex C++ programs developed and used by the experiments
  - "Simulation" and "Reconstruction" programs with hundreds or thousands of options to configure

Role of Python

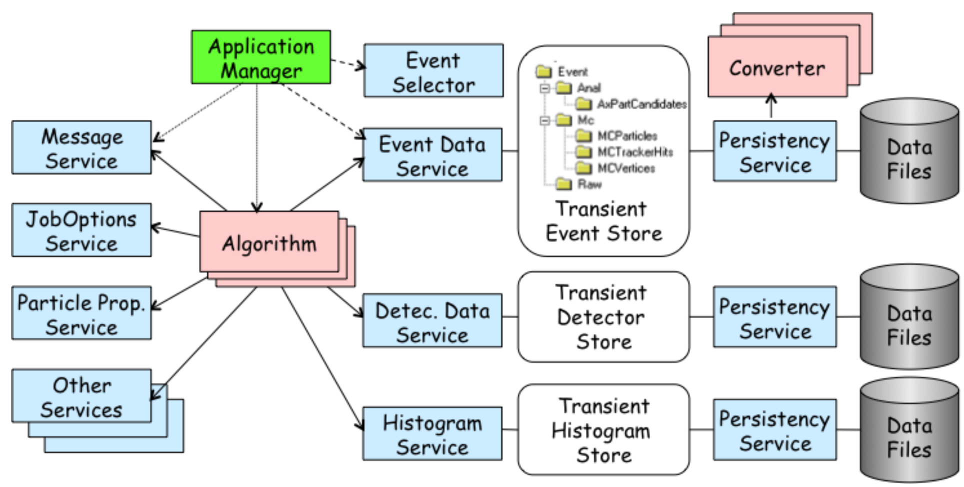
- Python language is really interesting for two main reasons:
  - High level programming language
    - Simple, elegant, easy to learn language
    - Ideal for rapid prototyping
    - Used for scientific programming (www.scipy.org)
  - Framework to “glue” different functionalities
    - Any two pieces of software can be glued at runtime if they offer a Python interface
    - With PyROOT any C++ class can be easy used from Python

Data Processing Frameworks

- Experiments have developed Software Frameworks
  - General architecture of any event processing applications (simulation, trigger, reconstruction, analysis, etc.)
  - To achieve coherency and to facilitate software re-use
  - Hide technical details to the end-user Physicists
  - Help the Physicists to focus on their physics algorithms
- Applications are developed by customizing the Framework
  - By the "composition" of elemental Algorithms to form complete applications
  - Using third-party components wherever possible and configuring them
- ALICE: AliROOT; ATLAS+LHCb: Athena/Gaudi; CMS: CMSSW

Example: GAUDI Framework

- GAUDI is a mature software framework for event data processing used by several HEP experiments
  - ATLAS, LHCb, HARP, GLAST, Daya Bay, Minerva, BES III,…
- The same framework is used for all applications
  - All applications behave the same way (configuration, logging, control, etc.)
  - Re-use of 'Services' (e.g. Det. description)
  - Re-use of 'Algorithms' (e.g. Recons -> HTL)
- Equivalent to MARLIN



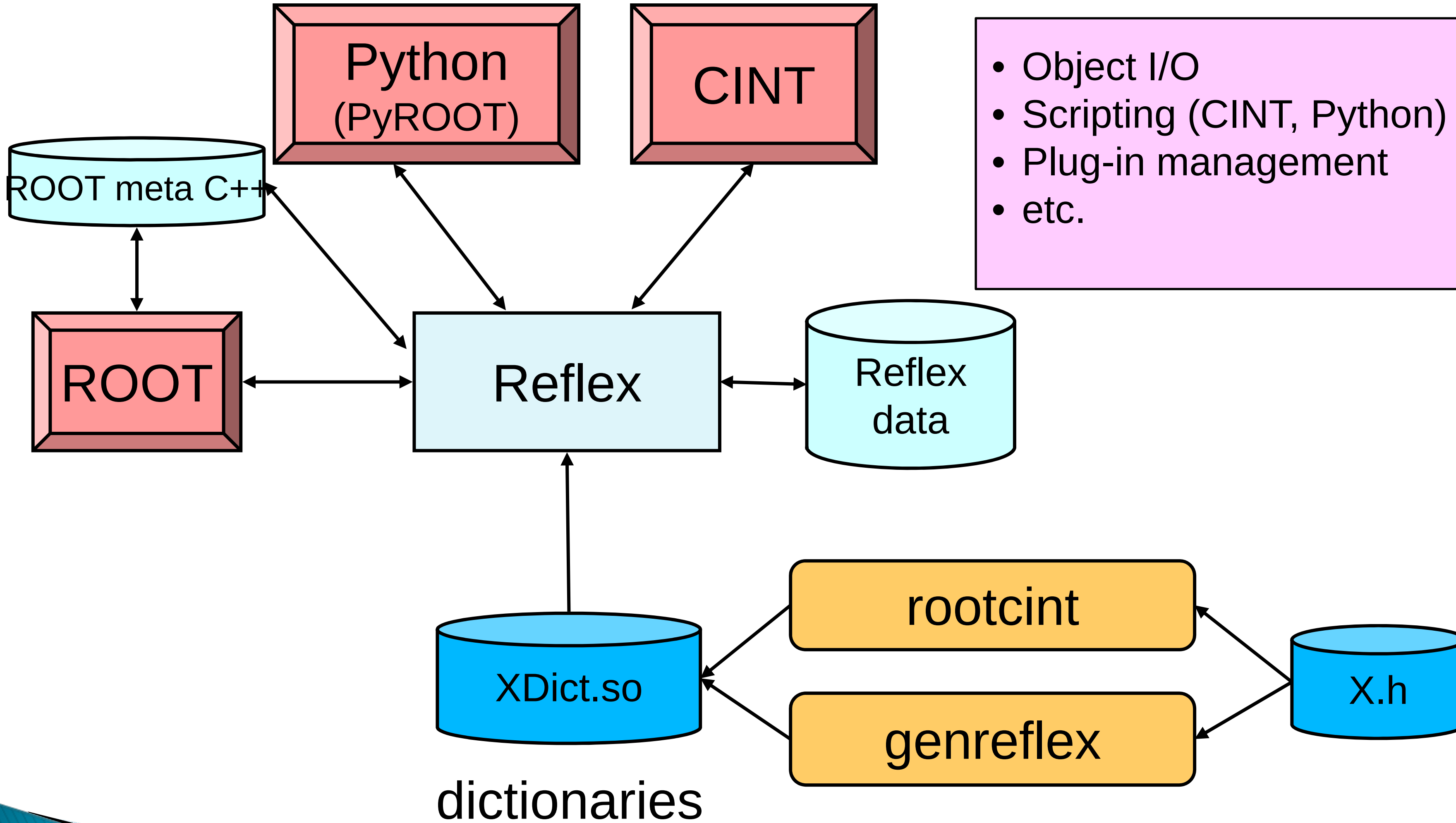
Application Interfaces

- Agreement on 'standard formats' for input and output data is essential for independent development of different detector concepts
  - Detector description, configuration data, MC generator data, raw data, reconstructed data, statistical data, etc.
- This has been the direction for the ILC software so far (e.g. lcdd, lcio, stdhep, gear,…)
- Unfortunately LHC has been standardizing on other formats/interfaces (e.g. HepMC, GDML, ROOT files, etc.)

Software Integrating Elements

- To facilitate the integration of independently developed components to build a coherent [single] program
- Dictionaries
  - Dictionaries provide meta data information (reflection) to allow introspection and interaction of objects in a generic manner
  - The LHC strategy has been a single reflection system (Reflex)
- Scripting languages
  - Interpreted languages are ideal for rapid prototyping
  - They allow integration of independently developed software modules (software bus)
  - Standardized on CINT and Python scripting languages
- Component model and plug-in management
  - Modeling the application as a set of components with well defined interfaces
  - Loading the required functionality at runtime

Strategic Role of C++ Reflexion



- Object I/O
- Scripting (CINT, Python)
- Plug-in management
- etc.

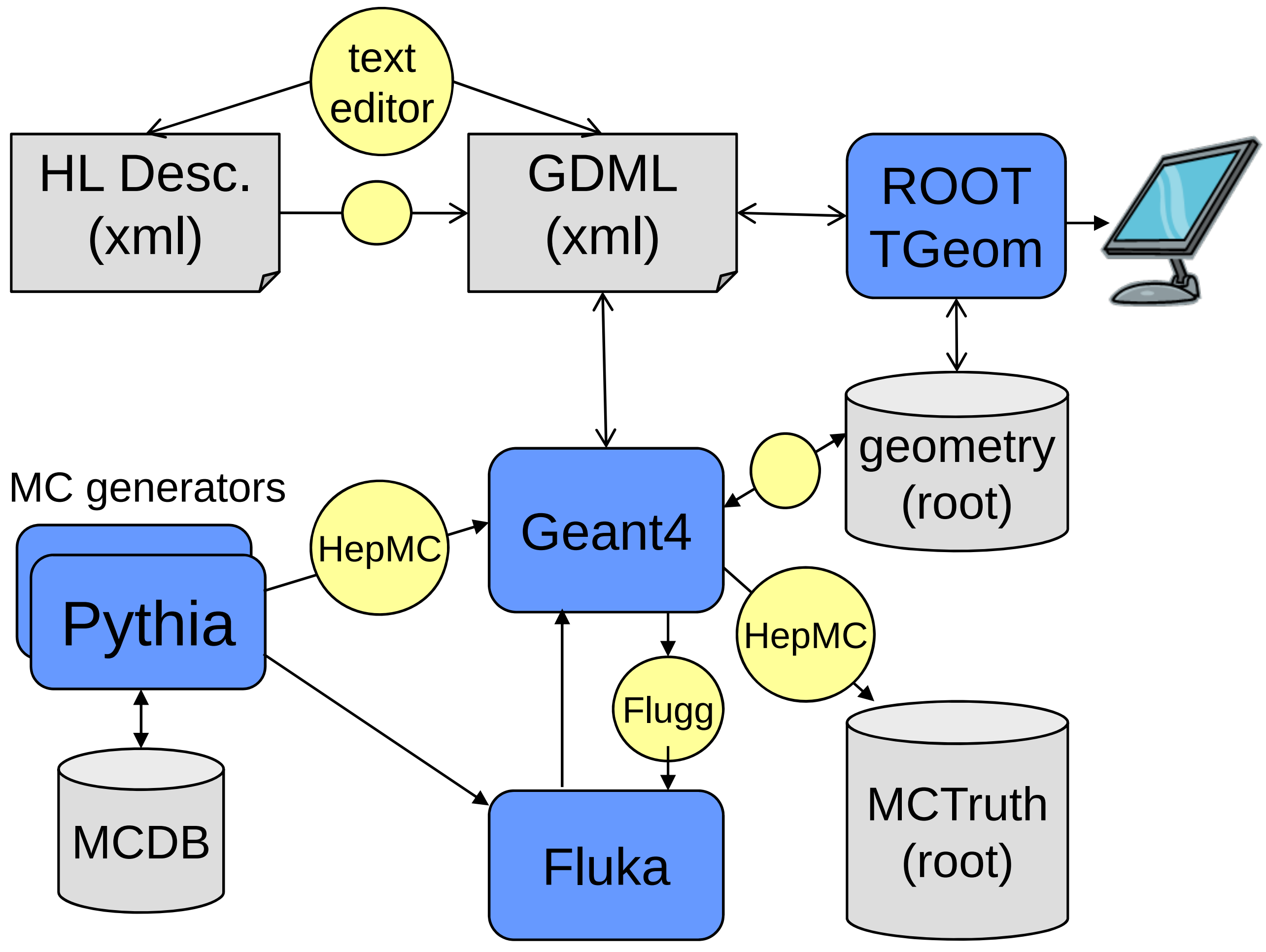
ROOT I/O

- Highly optimized (speed & size) platform independent I/O system developed for more than 10 years
  - Able to write/read any C++ object (event model independent)
  - Almost no restrictions (default constructor needed)
- Make use of ‘dictionaries’
- Self-describing files
  - Support for automatic and complex ‘schema evolution’
  - Usable without ‘user libraries’
- All the LHC experiments rely on ROOT I/O for the next many years

Data Persistency

- FILES – based on ROOT I/O
  - Complex data structure: event data, analysis data
  - Management of object relationships: file catalogues
  - Interface to Grid file catalogs and Grid file access
- Relational Databases – Oracle, MySQL, SQLite
  - Suitable for conditions, calibration, alignment, detector description data
  - Complex use cases difficult to be satisfied by a single solution
  - Isolating applications from the database implementations with a standardized relational database interface (CORAL)
    - facilitate the life of the developers
    - no change in the application to run in different environments
    - encode "good practices" once for all

Simulation Interfaces



GDML Exchange Format

- Geometry Description Markup Language (XML)
- Low level (materials, shapes, volumes and placements)
  - Quite verbose to edit directly
- Directly understood by Geant4 and ROOT
  - Long term commitment
- Has been extended by lcdd with sensitive volumes, regions, visualization attributes, etc.

HepMC Event Record

- Event record written in C++ for HEP MC Generators.
  - Many extensions from HEPEVT (the Fortran HEP standard common block)
- Agreed interface between MC authors and clients (LHC experiments, Geant4, …)
- I/O support
  - Read and Write ASCII files. Handy but not very efficient.
  - HepMC events can also be stored with ROOT I/O

Reconstruction Algorithms

- Unfortunately very little has been done in common between the LHC experiments
  - Different Event Models
  - Different Data Processing Frameworks
- The equivalent of the LCIO event data model did not exists
- Within a given experiment, alternative set of algorithms has been developed and could be evaluated thanks to the common event model and framework

Math, Statistical, Analysis,...

- ROOT is the facto standard and repository for all these common functionality
  - Histogramming, random numbers, liner algebra, numerical algorithms, fitting, multivariate analysis, statistical classes, etc.
- The aim is to continue to provide and support a coherent set of mathematical and statistical functions which are needed for the analysis, reconstruction and simulation

Software Configuration

- Re-using existing software packages saves on development effort but complicates "software configuration"
  - Need to hide this complexity
- A *configuration* is a combination of packages and versions that are coherent and compatible
- E.g. LHC experiments build their application software based on a given "LCG/AA configuration"
  - Interfaced to the experiments configuration systems (SCRAM, CMT)

Configuration of LCG software: LCG_40

Package: external

Version: 40

Platform: slc3_ia32_gcc323

Listing of configuration for LCG_lcg40

package	version
external	lcg40
gcc3	3.2.3
uuid	1.38
gccxml	0.6.0_patch3
CMake	1.8.3
boost	1.32.0_python242
bjam	3.1.10
python	2.4.2
clhep	1.9.2.2

Virtualization

- CernVM is a virtual appliance that is being developed to provide a portable and complete software environment for all LHC experiments
  - Decouples experiment software from system software and hardware
- It comes with a read only file system optimized for software distribution
  - Operational in offline mode for as long as you stay within the cache
  - Reduces the effort to install, maintain and keep up to date the experiment software

Summary

- LHC and ILC have been standardizing on different interfaces and packages
  - In some cases ILC is relying on packages considered ‘obsolete’ by the LHC community (e.g. CLHEP, AIDA, stdhep, …)
- Common interfaces/formats is good but adopting a common framework is even better
  - It would enable one level up in re-use
- ILC could leverage from existing structures and support for the common LHC software