Using ROOT classes from Python

ROOT Workshop

14-16 October 2002

P. Mato / CERN

Contents

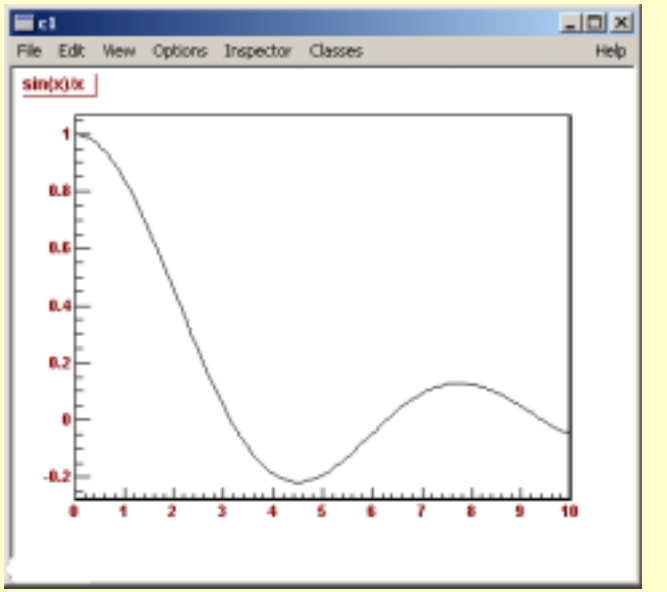
- Motivation
- Examples
- Why Python?
- Software requirements, Design, Status
- Integration using Python as a Software Bus
- Summary

Motivation

- Be able to use any ROOT class from Python in a generic way.
  - Without the need of wrapping each class
  - Using the ROOT object dictionary information
- Facilitate access of ROOT files and other facilities from non-ROOT applications
- Proof-of-concept that Python can be viewed as *Software Bus*
  - In analogy to a “hardware bus” where you can plug a variety of modules and interface adaptors to other buses.

Example

```
C:\> python
...
>>> from rootmodule import *
>>> f1 = TF1('func1','sin(x)/x',0,10)
>>> f1.Eval(3)
0.047040002686622402
>>> f1.Derivative(3)
-0.3456750567199266
>>> f1.Integral(0,3)
1.848652527994681
>>> f1.Draw()
<TCanvas::MakeDefCanvas>: created default TCanvas with name c1
```



- No much difference between CINT and Python !

Why Python?

- Interpreted
  - It is quite fast (byte code idea from Java)
- Dynamically typed
  - No need to declare any variable
- Simple syntax
  - Emphasis by the authors to minimize typing
- Variety of available shells
- Powerful built-in types and modules

  ➔ Ideal for Scripting and Prototyping
- Very popular nowadays (many users, many books(>20), many available modules (~1500), etc.)

Python Language

- Features:
  - Variables & Arithmetic expressions
  - String manipulations
  - Conditionals (if/else statements)
  - Loops (for/while)
  - Functions
  - Lists
  - Dictionaries
  - Classes (Objects)
  - Exceptions
  - Modules

Another example

makerootntuple.py

```
import sys, string
infile = open( 'aptuple.txt', 'r' )
lines  = infile.readlines()
title  = lines[0]
labels = string.split( lines[1] )

from rootmodule import TFile,TNtuple
outfile = TFile('aptuple.root','RECREATE',‘NTuple file')
ntuple  = TNtuple('ntuple', title, string.join(labels,':'))

for line in lines[2:]:
    words = string.split( line )
    row = map( float, words )
    apply( ntuple.Fill, row )

outfile.Write()
```

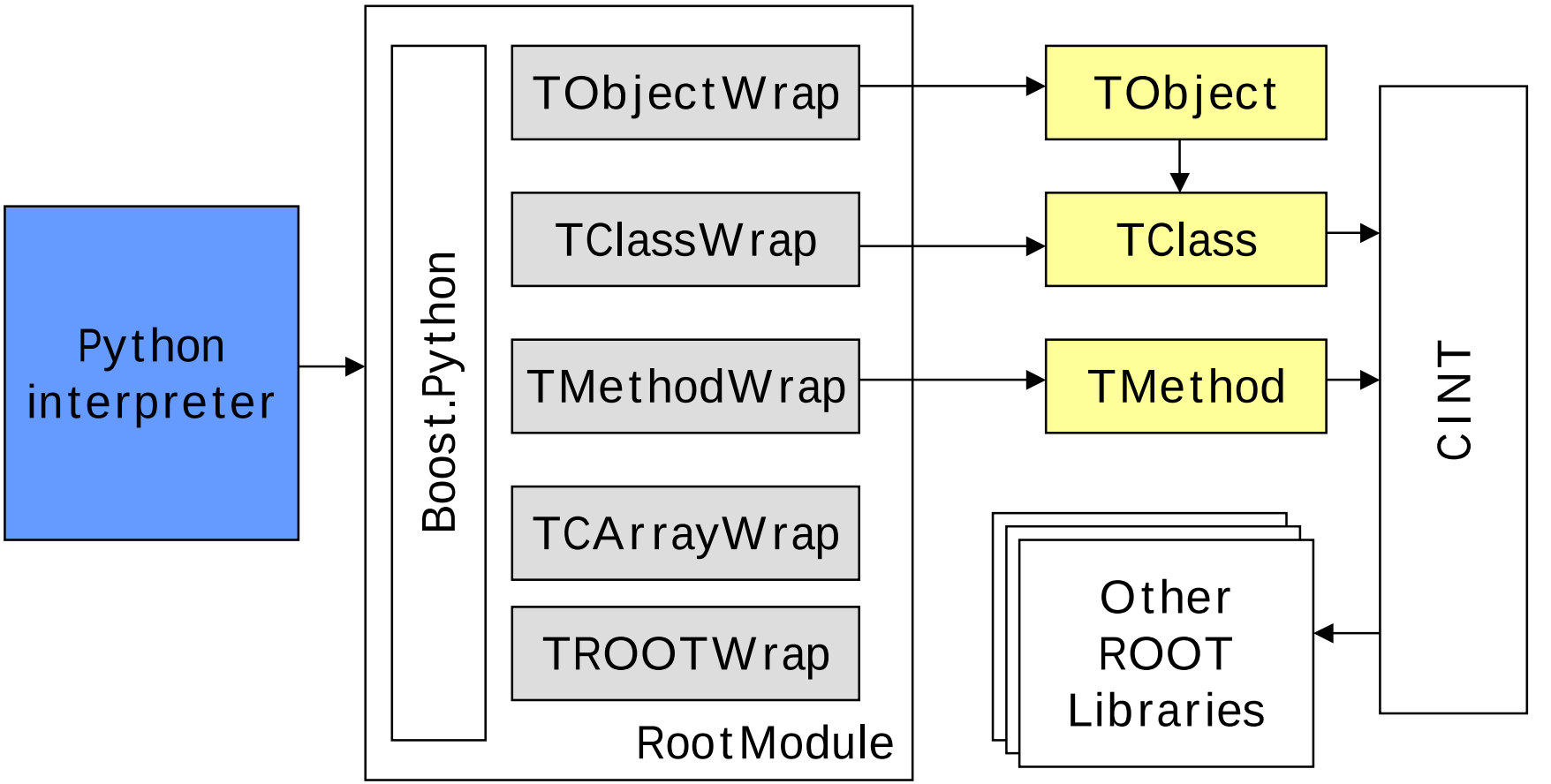
CERN Personnel
Category Division Flag Age Service …
202 9 15 58 28 0 10 13 4 40 11975
530 5 15 63 33 0 9 13 3 40 10228
316 9 15 56 31 2 9 13 7 40 10730
…

For each line in the file, split the line into a list of words. Convert the list of words into a list of floating-point numbers. And finally, call the Fill() method of the ntuple with it.

Software requirements

- Generic Interface
  - Make use of the generic python functions (__call__, __getattr__, …) to give *illusion* that all types are known to Python
  - Use ROOT dictionary information at run-time (perhaps sacrificing some performance). No need to generate any code/library.
- Emulating ROOT look and semantics while being Python natural
  - Same ROOT class names, method names, global variables, etc.
  - Python object instantiation, namespaces, garbage collection, etc.
- The “rootmodule” extension module should be self-contained
  - Nothing else should be required except *rootmodule.so[.dll]*

Design


Python interpreter
Boost.Python
TObjectWrap
TClassWrap
TMethodWrap
TCArrayWrap
TROOTWrap
RootModule
TObject
TClass
TMethod
CINT
Other ROOT Libraries

Boost.Python

- Boost.Python library helps exporting C++ classes to Python.
- It is designed to be minimally intrusive on your C++ design.
- You need to *reflect* the C++ classes and functions into Python

```
python::class_builder<TObjectWrap> tobject(this_module,
                                           "TObjectW");
tobject.def(&TObjectWrap::getValue,   "__getattr__");
tobject.def(&TObjectWrap::getName,    "GetName");
tobject.def(&TObjectWrap::className,  "ClassName");
tobject.def(&TObjectWrap::members,    "members");
tobject.def(&TObjectWrap::repr,       "__repr__");
tobject.def(&TObjectWrap::del,        "__del__");
```

C++ class/method

Python names

Boost.Python (cont'd)

- From the point of view of Python all classes are of type *TClassW*, all objects are of type *TObjectW,* all methods are of type *TMethodW*

```
>>> f1 = TF1('func1','sin(x)/x',0,10)
>>> dir(f1)
['ClassName', 'GetName', '__del__', '__doc__',
'__getattr__', '__module__', '__repr__', 'members']
>>> type(TF1)
<extension class rootmodule.TClassW at 794338>
>>> type(f1)
<extension class rootmodule.TObjectW at 78dbe0>
>>> type(f1.Draw)
<extension class rootmodule.TMethodW at 146c2c0>
```

Mapping ROOT from Python

- Types
  - Trivial mapping of basic types (float, string, …)
  - ROOT Classes into instances of TClassW python class
  - Not yet mapping of TCollection into python built-in types (list?, dictionary?).
- Global variables
  - Created instances with names gROOT, gSystem, etc.
- C-arrays
  - ROOT uses C arrays (e.g. float[], float*) and Python can not handle them. Created special type to handle it.

Known Problems

- The overall performance on calling methods is not optimal.
  - This is mainly due to the overheads in argument conversions. (Python types->C++ types->text string-> CINT interpretation->method)
  - Using the CINT dictionary directly should improve it.
- Calling python from CINT is not working if in different thread.
  - E.g. GUI call-backs on Windows
- Not all the basic types are probably supported yet. Specially of the "C" like types like arrays or pointers.

Known problems (cont'd)

- Naïve memory management (object ownership). ROOT objects created by Python are deleted by Python.
- The ROOT classes are not yet 100% equivalent to native Python classes, therefore some built-in functionality in shells (e.g. command completion) is not working.

It is available for testing

- The code can be downloaded from http://cern.ch/Gaudi/RootPython for tests
- It currently uses
  - Boost 1.27, Python 2.2, ROOT 3.03
- It is built using CMT for the time being
- Some python scripts examples exists at …/tutorials

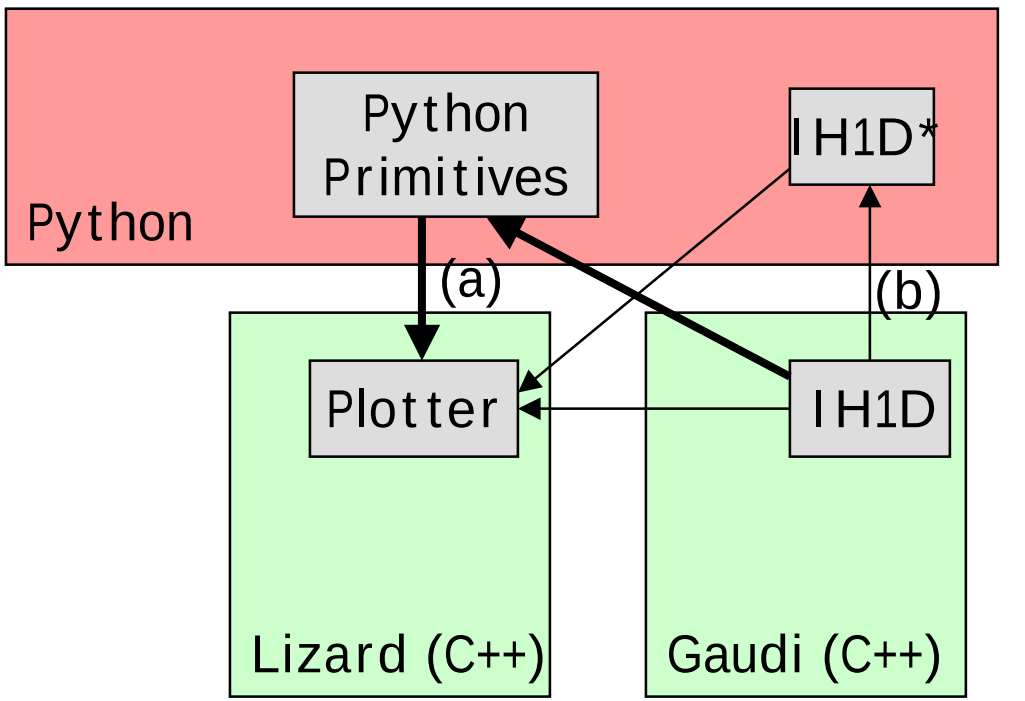
Python as a Software-Bus

- Python could also be seen as a framework where you can plug easily “extension modules” in binary form implemented using other languages.
  - Very easy and natural to interface to C++ classes (C++ API)
  - Python should only be the “glue” between modules developed in C++ or other languages
  - The interface (API) for Python extension modules is quite simple and at the same time very flexible (generic functions)

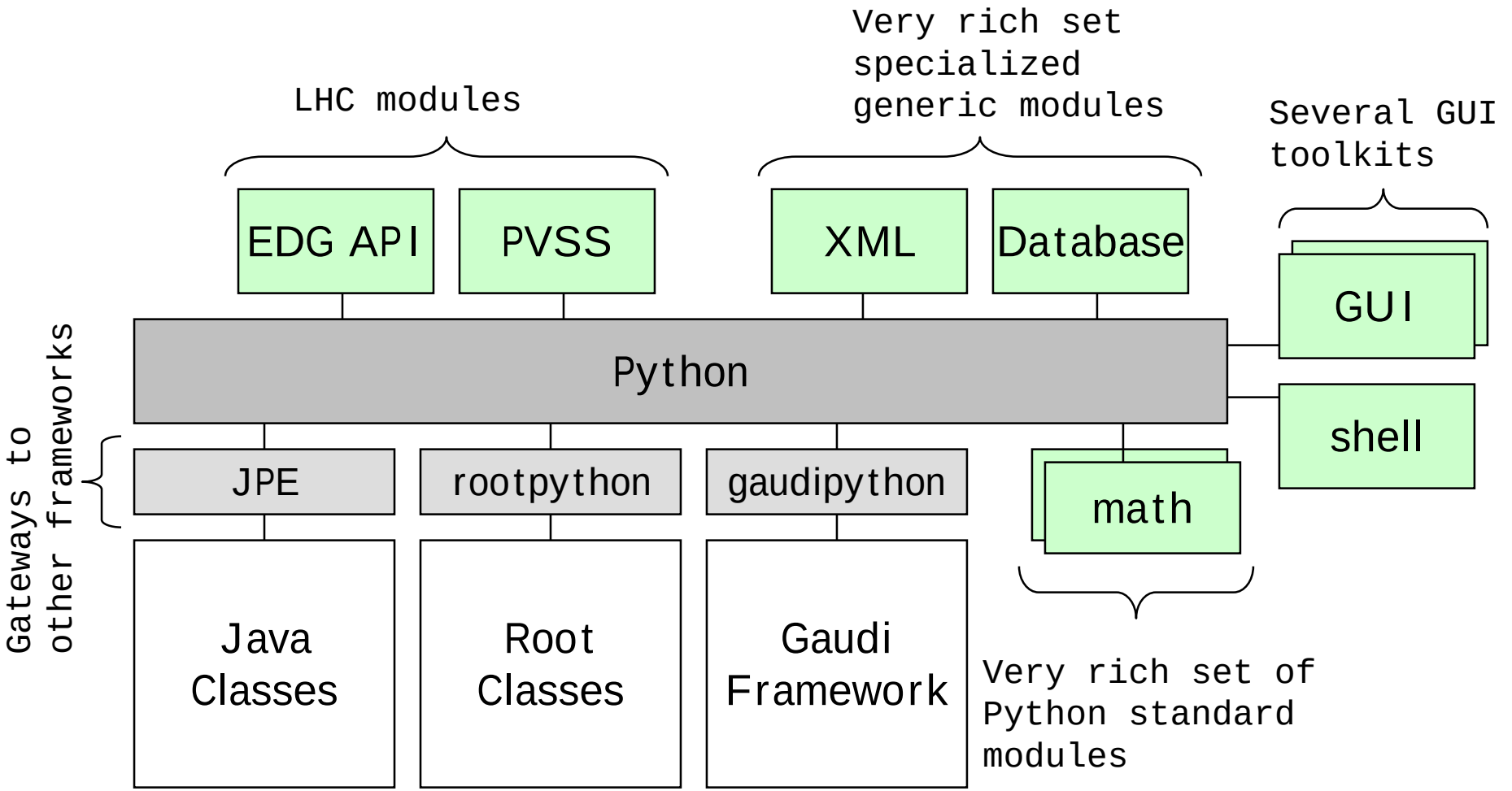
Module Communication

- Communication between two extension modules is always possible using Python primitive types (a)
  - Example: I would like to display an histogram of Gaudi
- It is more efficient to communicate using a reference to an interface (AIDA?) (b)
  - The objects in Python are only references

Both ways are possible!!



What is Feasible



Example

Filling an Excel spreadsheet from a ROOT ntuple

```
# Get the ntuple from the ROOT file
import rootmodule
hfile   = rootmodule.TFile('hsimple.root')
ntuple  = rootmodule.gROOT.FindObject('ntuple')
entries = ntuple.GetEntries()
nvar    = ntuple.GetNvar()
tuple   = ntuple.GetArgs()
# Initialize Excel
import win32com.client
excel  = win32com.client.Dispatch('Excel.Application')
wbook  = excel.Workbooks.Add()
wsheet = wbook.WorkSheets.Add()
wsheet.Name = ntuple.GetTitle()
# Fill Excel sheet
for i in xrange(500) :
  ntuple.GetEntry(i)
  for j in range(nvar) :
    wsheet.Cells(i+1,j+1).value = tuple[j]
# Make Excel sheet visible
excel.Visible = 1
```

Summary

- Exercise to proof that is possible to interface ROOT with Python with very little effort.
  - Demonstrate the power of having an object dictionary
- The module is functional and it is already in use to provide access to ROOT files from other applications (e.g. HippoDraw)
  - Of course, more work is needed to make it complete and fix the current performance problems
- Step in the direction of high level integration using a “software-bus”
  - One more Python “gateway” to other frameworks or languages like DCOM, Java, Gaudi, etc.