Pyragen

A PYTHON WRAPPER GENERATOR TO APPLICATION CORE LIBRARIES

EDMS tech note: https://edms.cern.ch/document/1343712/1
Software & manual: http://coflucluster/res/software/

Fernando PEREIRA, Christian THEIS - HSE/RP

Contents

- Motivation

  IPC framework? Objectives

- The framework

  Design overview, Entities, Rules and conventions

- How to
  - PyraGen Tool – IPC interface generator for Python
  - Preparing C++ applications: the ActiWiz case

Motivation

Web applications

Interactive front-ends

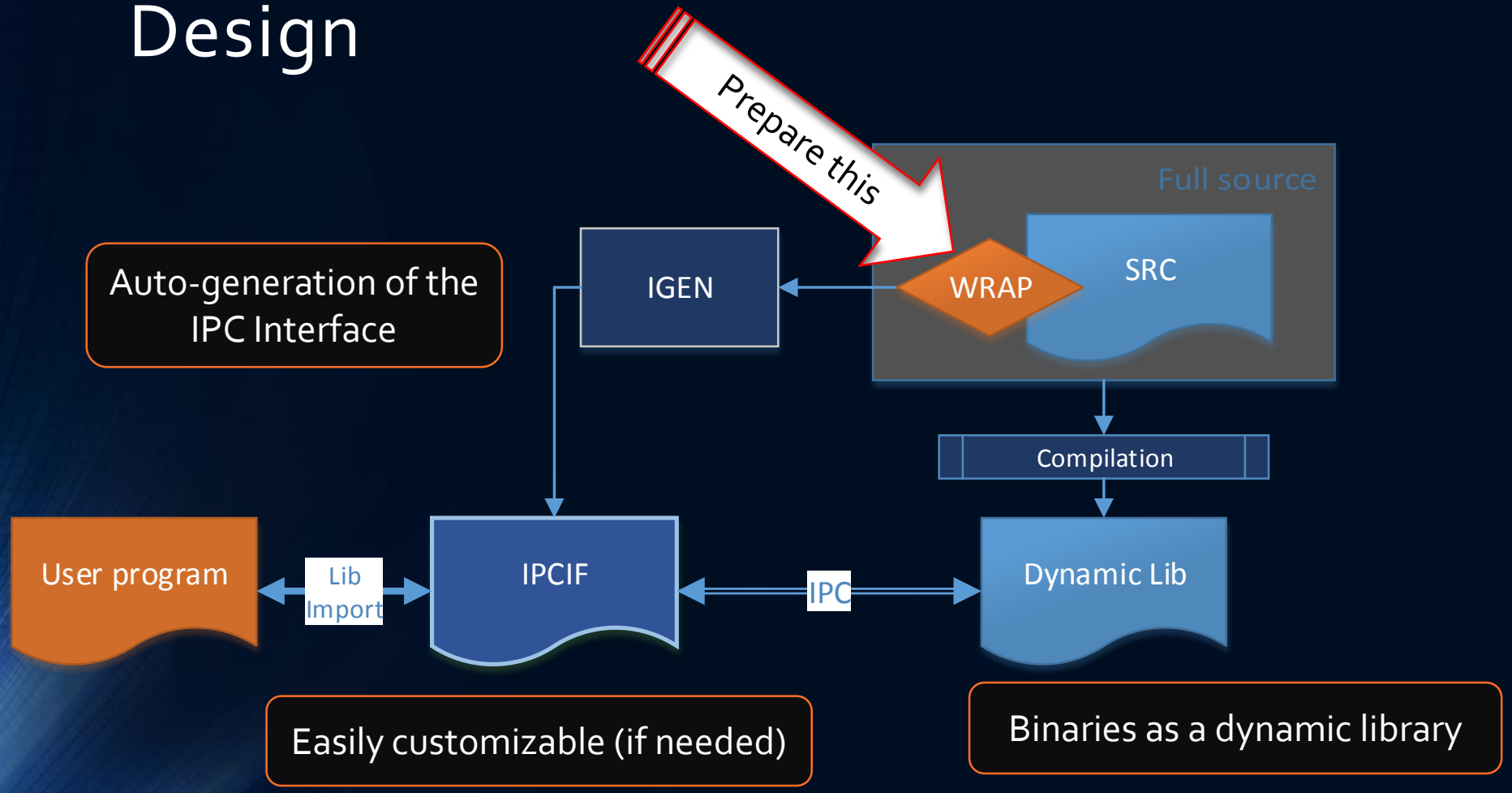
Complex code
(usually c++)

IPC framework

- Avoid re-development of existing well tested applications
  - Maximize investment in base code
  - Guaranteed consistent results
  - Save time and money!
- Do it simple, standard, customizable
  - Existing solutions tend to be too complex
    - not worth for simple tasks
    - Often require significant changes to the source code
  - A little bit on each side
    - A code interface in the core -Wrapper
    - A IPC caller interface in the target language (e.g. Python)

Code core

Design



1. Prepare WRAP
2. Compile full source to DLL
3. Run iGEN to create IPCIF
4. (Option) Customize IPCIF

Framework rules & conventions

- A function declared in the wrapper header → function in the IPCIF
  - Global method: No name change. Leading characters allowed.
  - Class constructor: Use of a trailing underscore after the class name.
    E.g.: ClassName * ClassName_();
  - Class destructor: Use of the trailing "_del" sequence after the class name.
    E.g.: void ClassName_del(ClassName* object);
  - Class member: Join class and method names with an underscore.
    E.g.: int ClassName_FunctionName(ClassName * object);
- Wrappers for functions returning complex results should encode them in JSON

*Automatic code generators require a precise naming convention. This is further strict in this case since actual function call are to be executed

Framework rules & conventions

- When implementing Object-Oriented style wrapper, functions must know which object they should refer to.

  *Therefore it is **mandatory** that*:

  - Constructor wrappers return the object pointer.
  - Methods accept a pointer to the object as first argument
  - Avoid name Clashing with existing functions. E.g.: prepend "I" (as for Interface)

  E.g.:

```
extern "C" PLUG_API
const char* IDecayEngine_FindInDecayChains( CDecayEngine* pPtr, const char* IsotopeId ) {
     pPtr->FindInDecayChains( IsotopeId ); // call original member function of the passed object
}
```

Tips to implement C++ wrapper – Actiwiz case

- Use standard C++ compliant & portable code only!
- Backend software (e.g. ActiWiz) needs to be compiled as a dynamic link library
- Working under Windows, Linux & Mac → use formalism shown in EDMS: 1343712 to keep one code base only
- All exposed functions must be specifically exported via a C specifier
- Wrapper functions can internally use objects (C++) but only C data-types can be exported

A Python Wrapper Generator - PyraGen

- Pyragen generates python IPC for C++ classes
  - by scanning API exposers/wrapper conforming to the proposed framework standard
- If offers a simple way to interface with C++ classes using **ctypes**
  - performance & flexibility
- Handles almost every type
  - all C base types (int, char, double) and combinations (e.g. unsigned short)
  - array pointers (strings as char *)
  - object pointers
- Creates clean code, easy to customize

Using PyraGen

```
# python pyragen.py
usage: pyragen source_header library_name [options]

optional arguments:
 -v   Verbose execution
```

```
# python pyragen.py PythonWrapper.h ActiWizCoreDyn.dll
Parsing PythonWrapper.h as header file
-> Processing function PyInitCore
-> Processing function IDecayEngine_
-> Processing function IDecayEngine_del
-> Processing function IDecayEngine_FindInDecayChains
(…)
Done Parsing.
Generating wrapper code to PyActiWizCoreDyn.py...
-> Wrapper class _Globals
(…)
-> Wrapper class INucProdSourceTerm
Complete.
```

Customizing IPC interface file

- Write custom code inside functions making the actual code.
- Call name/arguments **can't** be changed (they refer to functions in the DLL)
- Be very careful writing class code (e.g. new vars, more functionality to __init__), to not interfere with generated code

```
# (…)
class _Globals:
   #better not touch

#Wrapper for global function g_func_x
def g_func_x(MatName, Density):
  return _globalC._PyfuncX( MatName, Density )

(…)
#Please don't touch
#Init <=> module static method
_dll = ctypes.CDLL( './ActiWizCoreDyn.dll' )
_globalC = _Globals()
```

```
# (…)
class _Globals:
   #better not touch

#Wrapper for global function g_func_x
def g_func_x(MatName, Density, factor, max):
  new_density = Density * factor
  out = _globalC._PyfuncX( MatName, new_density )
  if out> max:
     return False
  else: return out
```

Example: use ActiWiz core to view a decay chain

Python code:

```
import NameOfLib as lib
res = lib.GetDecayChain("Cs-137")    # retrieve decay chain of Cs-137 and do a nicely formatted output

print json.dumps(res, sort_keys = False, indent = 4)
```

structured tree of all decay chains

Example 2: find all decay chains in which a specific isotope can be produced

Python code:

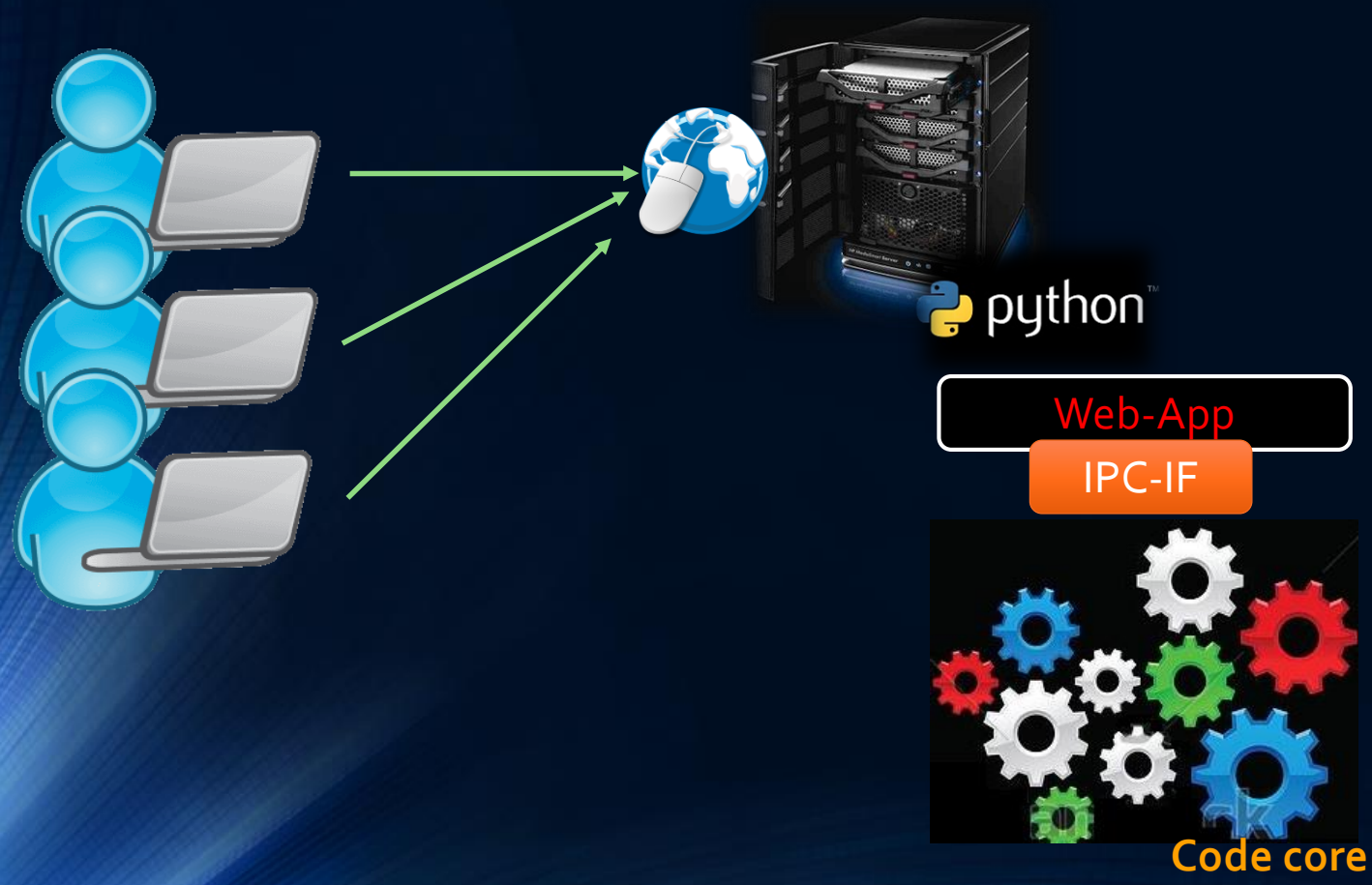
```
lib.InitCore()                          # initialize ActiWiz core library
engine = lib.DecayEngine()              # create decay engine object

res = engine.FindInDecayChains(“Ba-133")   # retrieve all chains which contain Ba-133

print json.dumps(res, sort_keys = False, indent = 4)
```

structured tree of all decay chains

Web-integration



- Web servers run python
  - Frameworks like DJANGO increasingly popular
    - Already used in several RP sites, e.g. ActiWeb, Radiolog, eSurvey
  - Python Web-Apps can use IPC interface like if it was the module itself
  - All the advantages of web are leveraged:
    - Updates / Maintenance is server side
    - No installation -> easy to try
    - Little resources in client

Thank you!

Questions?

- Contact us:
  - Fernando.Pereira@cern.ch
  - Christian.Theis@cern.ch