C++ as a service — rapid software development and dynamic interoperability with Python and beyond

Interactive C++: A Language InterOp Layer

Vassil Vassilev

07.10.2021

Motivation

- The C++ programming language is used for many numerically intensive scientific applications.
- C++ is often seen as difficult to learn and inconsistent with rapid application development
- The use of new programming languages has grown steadily in science and in fact Python is the language of choice for data science and application control but its computational performance is mediocre

Is there a way to combine the expressiveness of Python and the power of C++?

Classification of Prior Work

- Static binding generators — SWIG, SIP, Boost.Python, CxxWrap.jl
  They rely on limited parsers and limited type introspection to generate wrapper functions which the other language knows how to call. Advanced C++ features such as reference counting are hard to support.
- Dynamic / Automatic bindings — cppyy, Cxx.jl
  They operate on demand and generate only what's necessary when necessary. They rely on a compiler available at program runtime: cling and the Julia JIT respectively. They are efficient and can support about advanced C++ features such as template instantiation. Harder to implement.

How to define to whether two languages are interoperable?

Dynamic/Automatic bindings

- Move the binding provider responsibility from developer space to user space.

Performance Compared to Static Approaches

- No fundamental CPU performance difference

Note carefully that *everything* in Python is runtime: compile-time just means that the bindings *recipe* is compiled, not the actual bindings themselves!

- But heavy Cling/LLVM dependency:
  - ~25MB download cost; ~100MB memory overhead
  - Complex installation (and worse build)

Cppyy - Wim Lavrijsen, LBL, CaaS Monthly, Sep 2021

What is it?

- Interop package to combine C++/Julia in the same program
- One of two ways to combine Julia & C++
- One of the oldest Julia packages (> 8 years old originally targeted Julia 0.1)
- A C++ REPL environment
- A great proof of concept, that never got fully realized
- Disclaimer: Though I wrote most of the code, I haven't been maintaining it for > 3 years and haven't added new feature for > 6 years.

Cxx.jl - Keno Fischer, JuliaComputing, CaaS Monthly, Aug 2021

Goals

Create a C++ language interoperability layer allowing efficient dynamic/ automatic bindings with Python but also for D, Julia, etc…

- Create a document which describes prior art (cppyy and cxx.jl) and enumerates key features
- Implement a proof of concept which is able to instantiate a C++ template on the fly from within Python
- Rebase Cppyy (possibly Cxx.jl) on top the new implementation and measure efficiency

Definitions

For the purposes of the document we define:

- Introspection — the ability of the program to examine itself. The program should be able to answer the general question “What am I”?
- Reflection — the ability for a program to modify itself (including its behavior and its state).
- [Unqualified] Name lookup — the ability of the compiler to “find” by name internal objects representing C++ entities.
- Template instantiation — the ability of the compiler to produce a concrete entity from a template pattern (template<class T> T f(){} -> int f() {})

Template Instantiation on Demand

Dynamic tools which rely on clang can instantiate a template using the low-level libClang API. In Cling we can do:

```
[cling] struct S{};
[cling] cling::LookupHelper& LH = gCling->getLookupHelper()
(cling::LookupHelper &) @0x7fcba3c0bfc0
[cling] auto D = LH.findScope("std::vector<S>",
                              cling::LookupHelper::DiagSetting::NoDiagnostics)
(const clang::Decl *) 0x1216bdcd8
[cling] D->getDeclKindName()
(const char *) "ClassTemplateSpecialization"
```

Inefficient

Instable

C++ Name Lookup

In the following the following Python construct, the python interpreter will:

- Make a lookup for “val”, “std”, “vector” and “int”.
- Each unsuccessful name lookup will result in a callback which can introduce a name

Every unsuccessful lookup can be completed by a C++ entity connected to a python class wrapper.

```
val = std.vector[int]((1,2,3))
```

While parsing we can associate each construct with a C++ entity

C++ Templates

- It is challenging when supporting type systems that are less strict than C++
- Expression templates are challenging

int in python means integral type and it needs to be mapped to short, int, unsigned int…

```
val = std.vector[int]((1,2,3))
```

At the end of the statement, the interoperability layer will instantiate the C++ template std::vector<int> and initialize it with values 1,2,3

Overloads

When lookups return multiple candidates we need to implement a selection process:

- The default C++ overload resolution may not be best. (Eg. Python does not have a notion of const)
- Allow binding coded to handle overload resolution if required

Challenges

- Every language defines rules according to its design principles.
- An interoperability layer is a bridge between two languages as well as their design principles.
- The C++ type system has evolved over the years and has many performance-related aspects (eg const is part of overload resolution).
- How to define to whether two languages are **sufficiently** interoperable?

Proof of Concept

- Demo

Proposed Implementation

```
enum EntityKind {
  NamedDecl, TagDecl, NamespaceDecl, ...
};

using OpaqueCxxDecl = void*;

struct CxxEntity { // meant to expose parts of clang::Decl
  EntityKind Kind;
  constexpr auto name = "...";
  constexpr auto name_as_written = "...";
  constexpr auto value = "";
  CxxOpaqueDecl details; // pimpl
};

std::string getQualifiedNameAsString(OpaqueCxxDecl CxxD);
```

Using the pimpl pattern can help improve API stability

Proposed Implementation

```
using OpaqueCxxLookupResult = void*;
struct CxxLookupResult {
  std::vector<CxxEntity> Decls;
  void *details; // clang::LookupResult
};

CxxLookupResult R = cpp::lookup("std");
auto *StdNamespace = R.Decls[0];
CxxLookupResult R = cpp::lookup("vector", StdNamespace);

...
void DiagnoseAmbiguousOverloads(OpaqueCxxLookupResult R, ...);
```

Using the per-name lookups improve efficiency

Call For Action

- Many thanks to Wim Lavrijsen (LBL), Axel Naumann (CERN), David Lange (Princeton), Ioana Ifrim (Princeton), Bernhard Manfred Gruber (CERN, CASUS, TU Dresden) who contributed to earlier drafts of our work.
- Please take a look at the document and add comments. We aim to ‘release’ it next week.

Thank you!