If the shell is written in C++, why not just export its base classes?

devblogs.microsoft.com/oldnewthing/20111024-00

October 24, 2011

Raymond Chen

ton suggested that since the shell is written in C++, `IShellFolder` should have been an abstract class, and then it could have used techniques like exceptions and Inversion of Control.

Okay, first of all, I'm not sure how Inversion of Control is something that requires C++, so I'm going to leave that aside.

Second of all, who says the shell is written in C++? As it happens, when `IShellFolder` was introduced in Windows 95, the entire shell was written in plain C. That's right, plain C. Vtables were built up by hand, method inheritance was implemented by direct replacement in the vtable, method overrides were implemented by function chaining, multiple inheritance was implemented by manually moving the pointer around.

```
const IShellFolderVtbl c_vtblMyComputerSF =
{
 MyComputer_QueryInterfaceSF,
 MyComputer_AddRefSF,
 MyComputer_ReleaseSF,
 MyComputer_ParseDisplayName,
 ... you get the idea ...
};
const IPersistFolderVtbl c_vtblMyComputerPF =
{
 MyComputer_QueryInterfacePF,
 MyComputer_AddRefPF,
 MyComputer_ReleasePF,
 MyComputer_Initialize,
};
struct MyComputer {
 IShellFolder sf;
 IShellFolder pf;
 ULONG cRef;
 ... other member variables go here ...
}
MyComputer *MyComputer_New()
{
 MyComputer *self = malloc(sizeof(MyComputer));
 if (self) {
  self->sf.lpVtbl = &c_vtblMyComputerSF;
  self->pf.lpVtbl = &c_vtblMyComputerPF;
  self->cRef = 1;
  ... other "constructor" operations go here ...
 }
 return self;
}
// sample cast
MyComputer *pThis;
IPersistFolder *ppf =  &pThis->pf;
// sample method call
hr = IShellFolder_CompareIDs(psf, lParam, pidl1, pidl2);
// which expands to
hr = psf->lpVtbl->CompareIDs(psf, lParam, pidl1, pidl2);
// sample forwarder for multiply-derived method
HRESULT STDCALL MyComputer_QueryInterfacePF(
    IPersistFolder *selfPF, REFIID riid, void **ppv)
{
 MyComputer *self = CONTAINING_RECORD(selfPF, MyComputer, pf);
 return MyComputer_QueryInterfaceSF(&self->sf, riid, ppv);
}
```

So one good reason why the shell didn't export its C++ base classes was that *it didn't have any C++ base classes*.

Why choose C over C++? Well, at the time the Windows 95 project started, C++ was still a relatively new language for systems programming. While there were certainly people on the shell team capable of writing code in C++, the old-timers grew up with C as their native language, and the newcomers weren't taught C++ in their computer science classes. (Computer science departments still taught primarily C or Pascal, with maybe some Lisp if you took an AI class.) Also, the C++ compilers of the day did not provide fine control over automatic code generation,[1] and since even saving 4KB of memory had a perceptible impact on overall system performance, manually grouping rarely-used functions into the same region of memory of memory (so they could all remain paged out) was still a common practice.

But even if the shell was originally written in C++, exporting the base classes wouldn't have been a good idea. COM is a language-neutral platform. People have written COM objects in C, C++, Visual Basic, C#, Delphi, you-name-it. If `IShellFolder` were an exported C++ base class, then you have effectively said, "Sorry, only C++ code can implement `IShellFolder`. Screw off, all you other languages!"

But wait, it's worse than just that. Exporting a C++ base class ties you to a specific compiler vendor, because name decoration is not standardized. So it's not just "To implement `IShellFolder` you must use C++" but "To implement `IShellFolder` you must use the Microsoft Visual Studio C++ compiler."

But wait, it's worse than just that. The name decoration algorithm can even change between compiler versions. Furthermore, the mechanism by which exceptions are thrown and caught is not merely compiler-specific but compiler-*version* specific. If an exception is thrown by code compiled by one version of the C++ compiler and reaches code compiled by a different version of the C++ compiler, the results are undefined. (For example, the older version of the C++ compiler may not have supported RTTI.) So it's not just "To implement `IShellFolder` you must use C++" but "To implement `IShellFolder` you must use Microsoft Visual C++ 2.0." (So maybe Bjarne was right after all.)

But wait, it's worse than just that. Exporting a C++ base class means that the base class *can never change*, because various properties of the base class become hard-coded into the derived classes. The list of interfaces implemented by the base class becomes fixed. The size of the base class is fixed. Any inline methods are fixed. The precise layout of member variables is fixed. Exporting a C++ base class for `IShellFolder` would have meant that the base class could never change. You want support for `IShellFolder2`? Sorry, we can't add that without breaking everybody who compiled with the old header file.

Exercise: If exporting base classes is so horrible, why does the CLR do it all over the place?

Footnote

[1] Actually, I don't think even the C++ compilers of *today* give you fine control over automatic code generation, which is why Microsoft takes a conservative position on use of C++ in kernel mode, where the consequences of a poorly-timed page fault are much worse than simply poor performance. It will bluescreen your machine.

Raymond Chen

Follow