UCxx

The Improved C++ Binding for Babel

Jim Leek, Tom Epperly, & Gary Kumfert

Center for Applied Scientific Computing

January 27, 2005

This work was performed under the auspices of the U.S. Department of Energy by the University of California, Lawrence Livermore National Laboratory under Contract No. W-7405-Eng-48.
UCRL-PRES-209183

History

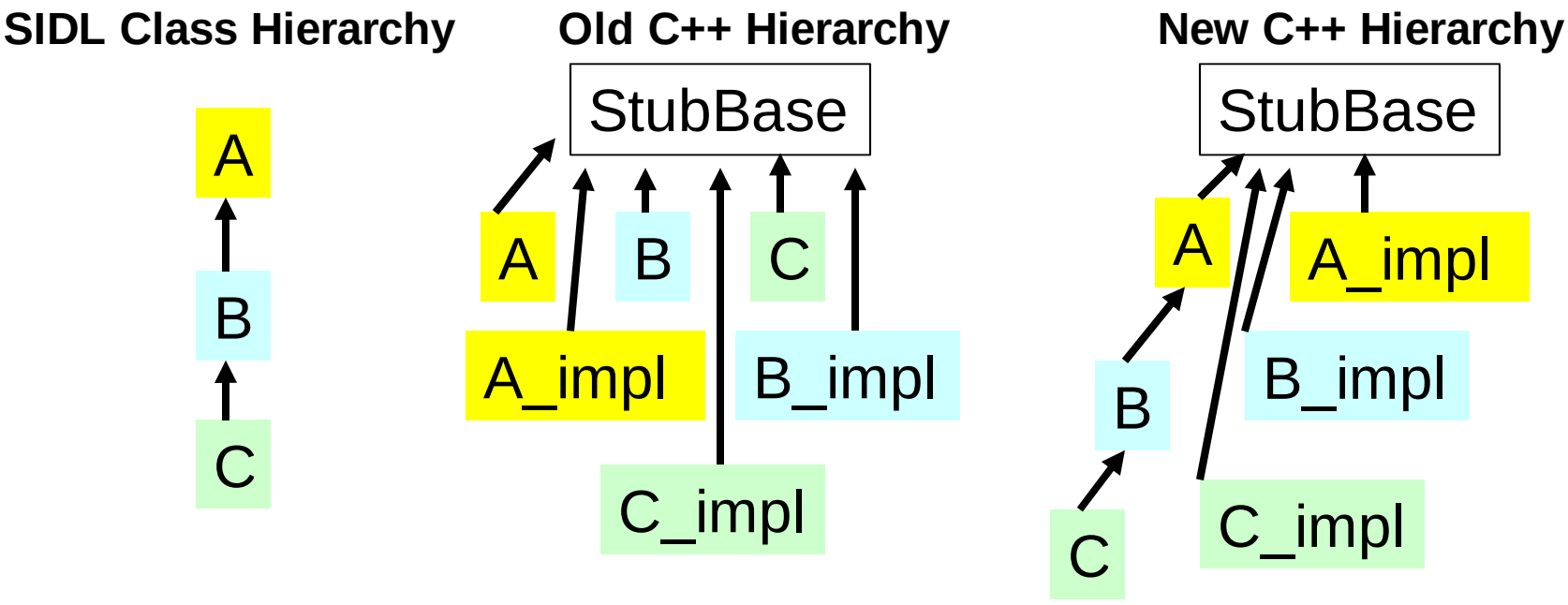
- It was the third age of mankind....
- The original Cxx was by Gary Kumfert
- As Babel evolved demand for new features grew
- Steve Parker prototyped what he wanted in a C++ binding for use with SCIRun.
- It was called U(tah)Cxx.

Goals

- O Implicit Upcasting
- O New babel_cast<>() operator for downcasts
- O Ability to call stub methods from the Impl without the self pointer.
- ~ Access from a derived Impl class to it's base Impl class's data.
- X C++ style throwing of derived class exceptions.

O Implicit Upcasting

- Implicit upcasting is simply a matter of reflecting the SIDL class hierarchy in the C++ class hierarchy.



O babel_cast<target>(source)

- The original Cxx binding overloaded the assignment operator to cast stubs. Now we use babel_cast. If the cast is bad, the result is nil.
- Old:

```
A a = return_c()
C c = a;
if(c._is_nil()) die();
```

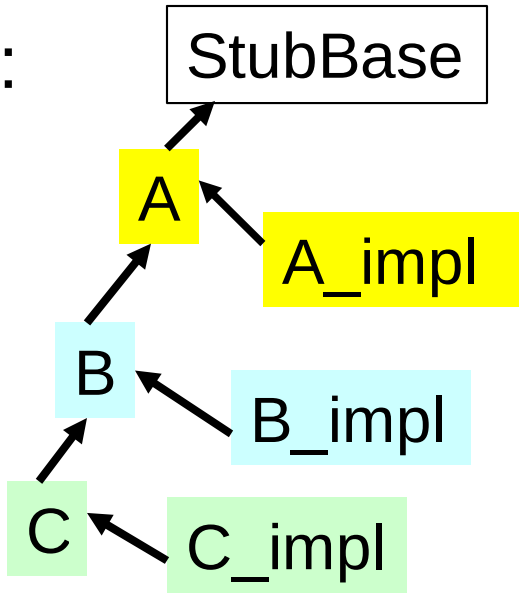
- New:

```
A a = return_c();
C c = babel_cast<C>(a);
if(c._is_nil()) die();
```

O Calling Stub from the Impl

- The Cxx binding included a “self” pointer for calling stub methods from the Impls
- UCxx we have the Impls inherit from the stubs so we can call directly.

Giving this hierarchy:



X Catching derived exceptions

- In C++ this is legal:

```
void foo() throw (A) { throw  C;}
int main() { try{ foo() } catch (C c) {/*…*/}}
```

- Something similar is possible with the Babel C binding.
- We received many requests to make this work with the UCxx binding. It doesn’t.

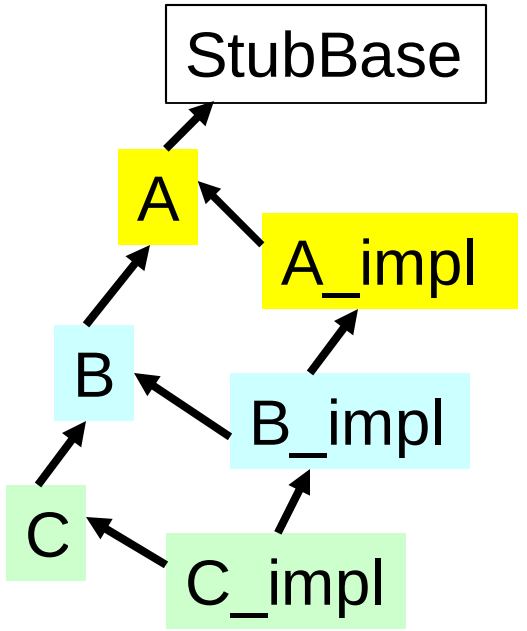
X It doesn’t work. Why not?

- It does not work in C++ because, with Babel, exceptions must pass through the IOR.
- It works in C because in C you catch the IOR pointer.
- It cannot work for C++ because the C++ binding must throw a type it expects to catch.
- The binding does, however, always attempt to match the most derived type first.

~ Access to a base Impl's data

- Many users requested that a derived Impl class be able to access it's base class's data directly.

- This immediately suggests some kind of inheritance. We decided on public.
- Giving us this hierarchy:

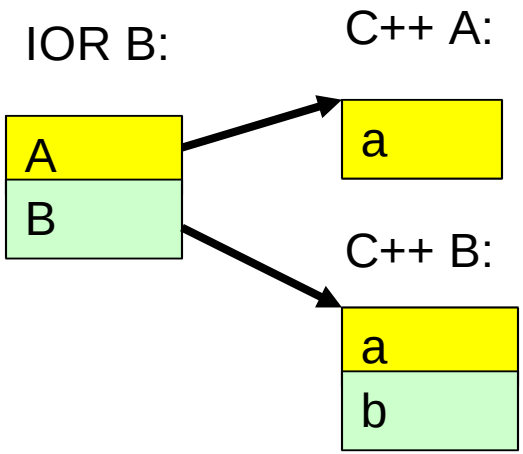


Which led to problems….

- First, not every Impl can reasonably be expected to access it's parent's data. What if the parent is written in Fortran?
- So such inheritance is optional. (The user must write it in the splicer blocks)
- Due to the possibility of diamond inheritance, all inheritance is now virtual.
- This will compile, the user may have his Impls inherit from each other. But…

It’s not the same data

- Unfortunately, this probably doesn’t do what you wanted. To see why, consider class B.



The IOR creates it’s own A for Babel inheritance! So you cannot access the A pointed to in the IOR from B.

Possible solutions?

- What might work to fix this problem?
  - Use placement new to initialize the C++ object with space allocated by the IOR.
  - Find some way to get the address of A_Impl and B_Impl as created by C++, and use them to initialize the IOR pointers.
  - Give the developer some way to get pointers to the IOR defined supers.
  - The developer could only export the top of the class hierarchy through Babel.

UCxx Downsides?

- I have not done a performance study yet, but I suspect (compared to Cxx):
  - Object creation/destruction may be a little slower
  - Function calls take the same amount of time
- All UCxx namespaces exist in the top level namespace ucxx. So:
  - ::ucxx::package1::package2::class
  - ::ucxx::sidl::array<bool> barray
  - (This is so Cxx and UCxx can be used together without collisions.)

Review
(What's new again?)

- The self pointer is gone from the Impls
  - Old: self.foo();
  - New: foo();
- Upcasting is implicit (bar takes an A)
  - Old: A a = c; bar(a);
  - New: bar(c);
- Downcasting uses babel_cast<>()
  - Old: C c = a;
  - New: C c = babel_cast<C>(a);

Tutorial Part 1

- This tutorial shows implicit upcasting and the babel_cast<>() operator.
- In this we have a Babelized priority queue that takes interface “Comparable.”
- We have a class “Integer” that implements Comparable.
- Put Integers into PriorityQueue and take them out again in order.

Tutorial Part 2

- In this tutorial we have a Babelized “Time Client.”
- A time client returns the time as a string. It has an interface for getting time from another machine over a network.
- “TimeClient” has a function “getTime” that makes a connection and gets the time.
- “TCPTimeClient” makes a tcp connection and gets the time from another machine.
- If getTime is called on a normal TimeClient, the time on the local machine is returned.

Conclusion

- UCxx makes the C++ Babel binding seem more like C++
- UCxx is still experimental, but we expect it to become the prefered C++ binding. However, for now details may change
- Ucxx also fulfills some of the Babel 1.0 release criteria.
- Please make suggestions about what you would like to see!