Subject: Writing an IDL application using a C++ data model.
Posted by Robbie on Mon, 12 Nov 2007 04:14:22 GMT
View Forum Message <> Reply to Message

Writing an IDL application using a C++ data model.

I was wondering if anyone has written an IDL application where most of
the data and meta data is stored and handled in C++?
Until now, I have taken the philosophy that I should call as little C+
+ as possible. In fact, all of my C++ is totally experimental is not
in production yet.

There are several reasons why I would prefer a C++ data model:
1) C++ is strongly typed, so the data model would be well defined.
2) C++ is much faster and flexible for organising meta data. IDL
structures are slow and tend towards disorganisation.
3) I could call external C++ code directly from the data model itself.

Here is an example class with a single property

```
class my_object: public idl_object {
public:
  float height;
  my_object() {
   height = 0;
  }
  float getHeight() {return(height);};
  void setHeight(float h) {height = h;};
};
```

The class could be registered in C++ as being accessible from IDL. I
would use C++ templates to assist in resolving type errors at compile
time.

```
registerClass<my_object>("MY_OBJECT");
registerProperty<float,
my_object>("HEIGHT",&my_object::getHeight,&my_object::setHeight);
```

The object could be instantiated and accessed from IDL like this:

```
obj = cpp_obj_new("MY_OBJECT")
obj -> GetProperty, HEIGHT=height
help, height
```

I have an existing implementation of casting variables from IDL to
boost::multi_array so the interface should support most IDL types.

Robbie

Subject: Re: Writing an IDL application using a C++ data model.
Posted by JD Smith on Mon, 12 Nov 2007 17:00:02 GMT
View Forum Message <> Reply to Message

> 2) C++ is much faster and flexible for organising meta data. IDL
> structures are slow and tend towards disorganisation.

Can you elaborate on this one with an example?

I do support the idea of a better "intrisic" method to extend IDL with
C/C++. MAKE_DLM is a step in that direction, but more could be done,
IMO, especially with the mapping between IDL types and native language
types.

There's no reason in this modern age that we don't have a big library of
extension DLM's which draw on the vast libraries of free numerical code
out there, other than the fact that DLM's are a) a bit fiddly to build, b)
challenging to use cross-platform. Just a little more wrapper/glue
assistance, along perhaps with a custom file type (.idlc ?) ala Matlab's
.mex files, would go a long way.

JD

Subject: Re: Writing an IDL application using a C++ data model.
Posted by Robbie on Mon, 12 Nov 2007 19:52:25 GMT
View Forum Message <> Reply to Message

Thanks for the reply. I don't envy whoever has to decide on the *best*
binding method.

>> 2) C++ is much faster and flexible for organising meta data. IDL
>> structures are slow and tend towards disorganisation.

It's a bit of a broad statement and I couldn't think about how to
narrow it down. I was actually referring to certain uses of structures
and objects.

A good example is that I have a set of heirarchical meta data. Nuclear
medicine files
have up to 8 dimensions, each representing detector head, energy
window, time sequence... and finally 3D spatial data.
It's far more convenient to simply represent the last 3 dimensions as
an associated variable and then use objects to categorise framesets of

data.

Many objects have vector properties. For example, a time sequence
object might have a 1D array FRAME_DURATION property.

The OO implementation gives me a chance to enforce constraints on the
property types. Because I'm using vectors, I end up with a structure
containing PTRs.

```
struct = {MY_OBJECT,$
  frame_duration: ptr_new()}
```

I think why bother with a structure at all? In fact I am currently
using an array of PTRs and array of strings because it is easier and
probably just as fast.

```
struct = {MY_OBJECT,$
  identifiers: ptr_new(), $
  values: ptr_new() $
}
```

At this point I wonder if I'm departing from a sensible usage of IDL.

I think IDL is a bit slow with method calls, particularily when I'm
looking for a child object with a particular property. A typical speed
up trick is to use a common variable as the backing store for that
property and search it using WHERE()

```
function my_object::findHeight, height
common MY_OBJECT, my_objects, my_object_height, my_object_parent
inds = where((height eq my_object_height) and (my_object_parent eq
self))
return, my_objects[inds[0]]
end

pro my_height::setProperty, HEIGHT=h
common MY_OBJECT, my_objects, my_object_height, my_object_parent
inds = where(self eq my_objects)
my_object_height[inds[0]] = h
end
```

So it seems that in order to get optimal performance from IDL, I must
fiddle with the backing store of objects. From my very limited
experience (very limited!) I suspect that I don't have to worry about
this is C++. I can rely on the standard library to sort this kind of
thing out for me.

Thanks

Robbie

Subject: Re: Writing an IDL application using a C++ data model.
Posted by David Fanning on Mon, 12 Nov 2007 20:09:45 GMT
View Forum Message <> Reply to Message

Robbie writes:

```
>  I think why bother with a structure at all? In fact I am currently
>  using an array of PTRs and array of strings because it is easier and
>  probably just as fast.
>
>  struct = {MY_OBJECT,$
>    identifiers: ptr_new(), $
>    values: ptr_new() $
>  }
>
>  At this point I wonder if I'm departing from a sensible usage of IDL.
>
>  I think IDL is a bit slow with method calls, particularily when I'm
>  looking for a child object with a particular property. A typical speed
>  up trick is to use a common variable as the backing store for that
>  property and search it using WHERE()
```

Here's my point. The '65 Mustang is cool and all that,
but you don't have to file the points and clean the carburetor
on the new model to get good performance. It just happens.

Why all the tricks? Why not just use a modern programming
language and be done with it? What is it about IDL that
keeps you so preoccupied with modifying it in all these
different ways?

Surely there is something better than IDL that meets your
needs. Or, isn't there?

I'm not trying to be antagonistic. I'm just curious.

Cheers,

David
--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Sepore ma de ni thui. ("Perhaps thou speakest truth.")