Subject: Re: Interactive Objects, Was: Simple GUI question
Posted by mvukovic on Thu, 24 Apr 2003 16:52:04 GMT
View Forum Message <> Reply to Message

David Fanning <david@dfanning.com> wrote in message
news:<MPG.191056febe370ae989b63@news.frii.com>...

... stuff deleted ...

> ... An image data object, to give you
> just one example, is particularly annoying. Should an image
> base class transparently handle 8-bit and 24-bit images? Or
> should these be two separate classes, primarily because they
> are handled differently in processing steps? We have been
> back and forth probably a thousand times on this one. It is
> probably one of the few times when we are *both* right!

... more stuff deleted ...

Here I would interject that the proper approach is to make an abstract image object with the basic interface defined but blank, except for common tools. Then make two objects (one for 8 and the other for 24 bit images) where the interfaces follow that of the abstract object. In addition, these two can extend the basic image object based on the additional properties of each image representation.

Now, strictly speaking, you do not need to make an abstract object in IDL, (I do it by making the init function return 0), but I like to do it to help me organize thoughts, and also spice-up debugging :-)

The big advantage I found was that once I define the interface and functionality using the abstract class, and then implement one object that inherits the base classs and test it in various applications, IT BECOMES MUCH EASIER (sorry for shouting) to implement a second object. It is almost guaranteed to work!

So, for example, first implement a simple ``_image'' object (I use preceeding undersocres for things that are supposed to be somewhat hidden) with many empty methods, but also with much of the pointers to data structures defined. It also contains some non-empty methods that would identically work for 8- and 24-bit images. Also, all the empty methods in _image print out a warning that they are not supposed to be called (they will be overriden shortly).

Next, implement the 8-bit image object that inherits _image, implements the various methods (and thus overrides the empty methods of _image), store the data in pointers, etc. Now you can test the 8-bit image object and start using it, and finding all the mistakes

that you made. On cleanup, don't forget to call the _image cleanup, and on init don't forget to first call _image's init.

So far, that was a lot of extra work. The real benefit comes when you come to implement the 24-bit image. You first take the 8-bit image object, and re-implement the routines that need re-implementing for the 24-bit image. The common procedures to both objects should go to _image.

But now, because the interface has been defined and tested beforehand on the 8-bit object, you are almost certain that when you start using the 24-bit object that it will work. That is the real benefit of this type of design. It leads to quite a bit of re-usability.

You can find all this and more in the ``pattern design'' book (if I recall the title correctly). That book helped me reach the object nirvana :-)

But, maybe you guys did consider that approach.

Mirko
