Subject: Re: Why IDL needs Garbage Collection
Posted by JD Smith on Thu, 21 Jul 2005 17:05:24 GMT
View Forum Message <> Reply to Message

Creating a reference counting super-class for all objects to inherit from isn't hard, and I suspect this is what David has done in Catalyst. The real problem is pointers, which have no such semantics.

One obvious option would be to make a composite pointer type which is actually an object, let's call it oPtr, which does nothing other than deliver the value of the contained pointer, and keep a reference count for it. A few problems with this:

1. There still is no automatic way to increment the reference count; programs would have to do this themselves whenever they store a copy of the oPtr object. If a program forgets, then the reference count is wrong and the pointer may be freed at the wrong time. Compare this to a system in which the program doesn't even have to think about reference counts, but instead just overwrite its pointers with impunity.

2. Programs still have to explicitly free all oPtr's when they are through with them (which doesn't always correspond to when they are through running). They can free things with impunity without worrying about stepping on someone else's toes, but they still have to actually free it. I.e., this type of system gaurds allows run-time flexibility, but not accommodate simple coding mistakes. Also, OBJ_DESTROY can't be used (since by the time the Cleanup code is called, it's too late). You'd have to introduce something like oPtr->Destroy (and make sure everyone uses that).

3. It's fairly heavy weight, and introduces an awkward syntax. Instead of *self.myptr, you'd need self.myoPtr->GetValue(). This creates a copy of the heap data, whereas *self.myptr simply references in the in-memory copy of the heap data. For small things, no big deal, for big things, potentially a very big deal.

So while this may appear to be a partial solution, I think coding garbage collection at a much lower level (essentially optimizing HEAP_GC, and having it run automatically) would be far superior. There are a wide variety of interesting GC technologies to choose among, from simple mark-and-sweep, reference counting, etc. to parallel algorithms which don't requiring stopping everything to do their work. Since IDL is now multi-threaded, it should be a manageable operation.

JD