Subject: Re: Why IDL needs Garbage Collection
Posted by Antonio Santiago on Thu, 21 Jul 2005 06:29:10 GMT
View Forum Message <> Reply to Message

I was thinking on the same problem a couple of days ago
(http://www.grahi.upc.edu/santiago/?p=149).

The implementation of IDL objects is pretty similary to the GObject
system.
GObject is a library written in C (as part of GTK+ project) that realizes
some object oriented compiler functions.

Both have an 'initializer' and a 'finalizer' object methods,
althought GObject has an 'initializer/finalizer' for the entery class.
Both use structure inheritance and only gives simple inheritance and
method overriding, but in GObject every time you create/destroy an object
the "system" controls the refcount of the object and only destryis it when
is 0.

Another point that surprise me, as C programmer, is the use of
"conventional memory" and "heap memory":
p=PTR_NEW('hello')

'p' is in "conventional memory" an is controlled by GC and 'hello' is in
heap memory and is my responsability free it.

a=10
p=PTR_NEW(a)

a and p are in "conventional memory" but 'p' points to a copy of 'a' at
heap memory. What?? I want a reference to the real 'a' !!!
Once you are familiarized with this it is no problem but I dont understant
the utility of this. I suposse it is for problems in object oriented
implementation and the pre-implemented "conventional memory".

Bye.

On Wed, 20 Jul 2005 14:24:43 -0700, JD Smith wrote:

>
> IDL pointers are great. We all use them to tuck things inside of
> structures, or pass around heavyweight data without penalty. IDL
> objects are great too, encapsulating data and functionality, enabling
> reasonably hassle-free GUI programming, and more. What is not great
> is the inflexibility that IDL's manual resource management imposes.
> Sure, object's have their Cleanup method, and that can be used to

> effectively free the object's heap data when the object is explicitly
> destroyed. Very useful. But, and this is the catch, that requires
> someone or something to continuously keep track of that object, and
> free it at the right time. Consider the simple case:
>
> IDL> a=obj_new('Foo')
> IDL> a=obj_new('Bar')
>
> Well, that's a memory leak right there. No one know about the 'Foo'
> object anymore. This is easy enough to avoid, but now imagine a
> system for passing around many many pointers and objects. For a
> concrete example, let's imagine a pointer pointing to a big pile of
> data called BOB. To keep from using too much memory, you don't want
> to replicate BOB in every corner of a set of applications that need to
> use it, so you allow different routines to share the BOB pointer.
> Fine. Well, what happens when a new BOB pointer gets sent in to
> occupy the same slot? Whose job is it to free the original BOB? How
> do you know someone else isn't still making use of the data being
> pointed to?
>
> Because of these types of issues, I find myself passing around lots of
> back-channel information like "make sure to free this pointer when you
> are done, but not this one, because I'll still be using that here,
> probably". Ugly. You can of course invent your own form of garbage
> collection (e.g. reference counting), but why shouldn't IDL, which
> clearly can keep track of heap data which is no longer being pointed
> to (vz. HEAP_GC,/VERBOSE), do the dirty work for you? Then, whether a
> pointer or object is shared across 10 different programs for the
> duration of an IDL session, or simply created, used once, and then
> discarded, you wouldn't need any additional logic to decide if and
> when to free a given resource. And no, I don't consider putting
> HEAP_GC in your event callback effective garbage collection.
>
> This is why RSI needs to implement a simple but effective garbage
> collection paradigm in the next version of IDL. Anyone agree?
>
> JD

--

Antonio Santiago Pï¿½rez
(email: santiago<<at>>grahi.upc.edu)
(www: http://www.grahi.upc.edu/santiago)
(www: http://asantiago.blogsite.org)

GRAHI - Grup de Recerca Aplicada en Hidrometeorologia
Universitat Politï¿½cnica de Catalunya
