Subject: Re: passing parameters from base to base
Posted by JD Smith on Mon, 02 Dec 2002 20:47:29 GMT
View Forum Message <> Reply to Message

On Mon, 02 Dec 2002 09:49:31 -0700, Pavel A. Romashkin wrote:

> Aha, I knew it! :-)
> Yes, I think this is adequate. And what David is talking about seems
> pretty much the same and Ok too, as long as traditional relationships
> among programs are preserved, i.e. one is launched from another, or a
> known/predictable sequence of launches exists. As long as one can/wants
> to *pass* anything to another process, everything seems trivial, and
> messaging system (as it appears to me) is a neat way of inter-object
> communication. I faced this too but what I wanted is absolutely no
> assumptions regarding what will launch when, and I did not want to keep
> any notations of what *has* already launched. It should all happen by
> itself. As long as one is allowed to have a Common variable, this is
> simple to achieve because you have a place to keep your "watchdog"
> object, but if you wanted to get by without Commons, it is more
> difficult and is not nearly as neat anymore.

The relationships don't need to be the same; in fact, accomodating new code without re-writing the original was the foremost and essential idea of this setup. I can write an entirely different object I hadn't thought of at the time, and link it simply using the published messages (or add another message type if what I need isn't available). It's really a compile-time vs. run-time, *not* a code-time vs. run-time static communication pattern issue. If the latter were the case, there'd be no reason to use objects at all.

You may be confusing the system I describe as one for widget events primarily, with the attendant notion of "launching" that widget vis. a vis XManager. I have ObjMsg objects which have no widget component whatsoever. And ones which do have a widget component typically are introspective: if the widget doesn't exist when it's needed, it's created.

> P.S. BTW, I think that "relations and introductions can be done at
> compile time" is an overstatement. Objects have to exist in order to
> link anything to them, because their location in heap memory is not
> known until then.

Yes, you have to watch your ordering. You'll notice my oDraw was instantiated before the others which depend on it. When this is impossible, a run-time class or name-based lookup can solve this (as with GetMsgObjs()). Another technique which mitigates this problem in practice is object composition: if one object has an single-point dependency on another, the simplest way to organize them is to have

the primary object create its very own instance of the subsidiary (at which point it is free to setup the relevant communication channels).

If it's a total free-for-all of bizarre interdependencies, you almost *have* to establish the channels of communication at run time. Of course, then you're not at all guaranteed that the conversation won't devolve into name calling and grunting ;). The ultimate limit of this is to search for objects to speak with everytime you have something to say, or something you'd like to hear about. Imagine that what you are saying or listening for is a motion event, and you'll quickly see this isn't a good idea ;).

JD

> JD Smith wrote:
>>
>> and oFunk could sign up for the tvDraw messages it needs. Perhaps
>> oFunk even builds an ObjMsg object of its own and signs up with it
>> directly. Usually this is perfectly adequate: relations and
>> introductions can be done at compile time or Init time. This works
>> well for, e.g., building a viewer shell with all the desired plug-ins,
>> especially when you need to carry widget bases around to build the
>> interface inside.
>>
>>
> ... snip ...
>
>
>> Any suggestions much appreciated. Another option is to publish
>> "capabilities" or lists of potential messages sent in a common block of
>> some kind, but that gets complicated with many multiples of the same
>> type of object on the global stack.