Subject: Re: passing parameters from base to base
Posted by JD Smith on Tue, 03 Dec 2002 20:10:27 GMT
View Forum Message <> Reply to Message

On Tue, 03 Dec 2002 10:14:52 -0700, Stein Vidar Hagfors Haugan wrote:

```
>  "Pavel A. Romashkin" <pavel_romashkin@hotmail.com> writes:
>
>>  JD Smith wrote:
>>>
>>>  In an ideal system, ObjMsg objects would find each other and setup
>>>  their own intercommunication themselves.  You get into lots of
>>>  chicken-and-egg dependency issues in this case though.
>>
>>  This is exactly what I was talking about :-)
>
>  I may not be getting exactly what you're talking about, but this is
>  exactly where I think a singleton might be useful: You can always "find"
>  it (it'll be created & initialized if it doesn't exist!) as long as you
>  know it's name, e.g. (pardon any mis-programming here, I haven't been
>  keeping up with objects in daily life!)
>
>     ;; Locate the message center
>     dummy = obj_new("message_center",object=MsgCtr)
>
>     MsgCtr->Register,self,"mainprog",uniqkey  ;; Key allows multiple
>     instances
>                                               ;; of mainprog
>
>     ;; The LocateObjMsg could generate a proxy object if ObjMsg ;;
>     hasn't registered yet!  The key makes sure we don't pick a ;; wrong
>     instance of ObjMsg (that has already been connected ;; with another
>     self-class object)
>
>     objmsg = MsgCtr->LocateObjMsg("ObjMsg",uniqkey)
>
>     if objmsg->isproxy() then begin
>        objmsg->queue_state,/on            ;; Keep all messages I send
>        objmsg->inform_me_when_registered,self,"through_this_method "
>     end
>
>     objmsg->send_message,self,"Send me data
>     anytime","through_this_method"
>
>  Etc, etc.. Now, when "ObjMsg" registers with the message center, it
>  (MsgCtr) will generate a call to the above object's
>  "through_this_method" method, with information that the proxy "objmsg"
>  is now replaced with a true ObjMsg instance, with so-and-so unique key.
```

> It will then dump all the queued messages from the proxy onto the
> registering ObjMsg (maybe triggered by ObjMsg calling a
> MsgCtr->deliver_pending_messages method, gives you more control over
> whether or not the object is fully initialized).
>
> The number of variations on this scheme is endless, and details will
> differ according to what kind of messaging system is already
> implemented. However, it sketches out a method to deal with
> chicken-and-egg problems.
>
> One key problem might be how to use the unique keys (e.g. system time at
> object creation) to enable creation of multiple independent interlinked
> systems (e.g., two display windows with separate sets of linked
> controls). Only a half-thought-through solution, but I don't see that
> you have problems that cannot be solved by this general approach

I see what you mean: using a singleton as sort of the grand
matchmaker, for a more organized system for setting up communication.

Really the ObjMsg framework is already designed to do much of this for
you (delivering messages to the correct place at the correct time).
The major differences between your proposed approach and mine are
worth highlighting:

- The ObjMsg system relies on each individual object which is
  interested in communicating with the world to maintain its own
  personal list of message recipients, and to find other ObjMsg
  objects, and sign up for the messages it wants to hear about from
  them. The latter can be accomplished in a variety of ways: by
  direct "introduction" (often at a higher level where the objects
  are created), by "composition" of subsidiary objects, and by
  "friend of a friend" introductions. One other subtlety: *all*
  messages are delivered through a single "Message" method.

- The MsgCtr system centralizes *all* messaging objects, and allows
  you to find any one of them using a class name and unique ID.
  Presumably, it's a unique ID you already know, or you're back in
  the same boat. Messaging objects can sign up with it and reveal
  how they can be contacted. I'm not sure how objects specifically
  request messages, but nonetheless...

The biggest problem I see with a centralized system is the "dependency
hell" so familiar to anyone who does package management. Suppose I
have 5 instances of a viewer running. Each little plug-in tool of the
viewer is a separate object which needs to contact a 'tvDraw' object
to get draw events. In your system, how does it go about doing that?
Well, if each tvDraw object had a unique ID, that would work. But how
do you distribute all those unique ID's?... and you're back in the same

boat again.

Thanks for the suggestions.

JD
