Subject: Re: Passing info and destroying widgets...
Posted by J.D. Smith on Thu, 24 Jun 1999 07:00:00 GMT
View Forum Message <> Reply to Message

Struan Gray wrote:

> The idea of a objectified widget I owe to Mark Rivers. Deja News
> has a thead with a neat discussion of his technique, plus a few
> refinements - search on his name and 'objects'. My widgets follow his
> scheme, with a few inherited properties that I like all my
> program-oriented objects have (such as a unified way of handling
> global and user preferences) and generalised information
> sharing/passing methods (the above, plus the ability to handle
> conventional events).
>
> At present the parts work, but the whole looks like it's in the
> middle of open heart surgery. I'm building a disperse set of
> data-objects, widget-objects and plot/analysis-objects and at present
> I'm playing around with different ways of distributing basic
> behaviours among them. I'm not sure when it will be ready for public
> consumption, but I promise to make what I have freely available when
> it is.

I have developed something along these lines and have been using it for over a year in all my new widget applications. I'd be interested in seeing what you have and how it compares. I really think object widgets (Obgets, for short) are the way to go; so much so, in fact, that I've toyed with rewriting XManager to be more object friendly. No reason I shouldn't be able to say:

XManager,"MyApp:"+self.name, self.base, /NO_BLOCK, OBJECT=self, Method='thisEventHandler'

But even this doesn't go far enough. Taking this progression even further, XManager could be scrapped altogether in favor of an event-handling, message passing class for obgets. Though my framework, contained in a class called "ObjMsg", does not directly manage the widget events of its obget progeny (requiring XMananger still to be used), it does implement a generic, feature-based message passing protocol, which has proven very useful. ObjMsg objects can "publicize" the features or services they offer, and others can "subscribe" to those services, all of which can be modified dynamically during runtime. Some of these features might be just the passing of plain widget events being generated in a widget an obget contains, but more commonly, they are messages distilled from one or more widget events, or independent of widget events altogether. Here's the first paragraph of the doc/blurb:

;+

; NAME: ObjMsg
;
; PURPOSE: A superclass to define a common prescription for event-
;   driven object communication. The events will include those
;   which arise from widget activity within the objects, but the
;   formalism is extensible to any generic 'object events'. Both
;   kinds of events are encapsulated by the term 'messages', and
;   are referred to as "object messages" when handled by the
;   protocol defined in this class.

Basically, once you have a generic message passing protocol, you are free to implement whatever message flow structure is convenient, not just the "up-the-widget-tree" technique implicit in normal widget programming.

With this freedom comes complexity, but also power. For instance, one nice feature is the ability to debug your large application by "snooping" on the messages being passed. Since they all travel through the ObjMsg class, you can for instance, do things like "show all messages of types [msg1,mgs2,msg3] from objects of type X, and show who is sending them to whom, and the calling sequence which resulted in their being sent".

I would like to make several improvements, and I think some kind of obget programming super-class could be very useful, potentially even part of the IDL distribution. Things I would like to do include:

1. Handle the widget events of the obget progeny -- i.e. obviate XManager. While XManager is a "private" RSI routine, it really doesn't do too much at all, just calling widget_info, widget_control, and widget_event, albeit with a few undocumented keywords. ObjMsg wouldn't have to provide backward compatibility for "fake" Modal widgets, and non-blocking widgets aren't really handled by XManager anyway, so this changeover could be accomplished without too much fuss.

2. Make the service "publication" and "subscription" more robust, uniform, and easier to understand. Currently each ObjMsg object is responsible for organizing it's own services for publication, which leads to different conventions for subscription to different class services. Ideally, anyone could write an class which inherits ObjMsg, name and describe the services it provides, and, without seeing any source code, someone else could subscribe to and make use of those services.

3. Update the debugging features mentioned above.

Anyway, it seems like quite a few of you are converging on this type of idea, and I bet we could come up with something very useable if we put

our heads together.

JD

--

J.D. Smith |*| WORK: (607) 255-5842
Cornell University Dept. of Astronomy |*| (607) 255-6263
304 Space Sciences Bldg. |*| FAX: (607) 255-5875
Ithaca, NY 14853 |*|