OriginalTweakMemo

To: "Alan Kay", "Dan Ingalls", "John Maloney", "Kim Rose", "Michael Rueger", "Scott Wallace", "Ted Kaehler", "Bob Arning"

Subject: Events, Scripts & Multiple Processes

Date: Fri, 6 Jul 2001 22:07:01 -0800

Folks,

I've done some thinking about events and scripts and it appears to me that we're overlooking a few fundamental issues here. As an example, when Alan talks about "events within scripts" I think it is important to understand that the most common use of it will not be some sort of #firstTime:eachTime:lastTime: based on some mouse click behavior but much rather a dog-simple use along the lines of:

```
        button label: 'Click to play movie'.
        button waitUntil: #click.
        movie startPlaing.
        movie waitUntil: #done.
```

The immanent problem in doing anything like the above (or using any kinds of events from within methods) is that in such an environment (or any media system in general) everything is about time. If you write a media-script you need to deal with time and you need to do so in a straightforward way. E.g., the above is obviously a simple and desirable way of doing things but in the current system the above would require us to use multiple indirections (such as using #on:send:to:). An even worse example is "Alan's slide sequencer" which should be expressed by:

```
        slide1 show.
        nextButton waitUntil: #click.
        slide2 show.
        nextButton waitUntil: #click.
        slide3 show.
```

This not only requires the indirection(s) from the above but in addition we even need to stitch the sequence back together! (Alan does this by using an extra variable) What a horrible construct to explain to an omni-user ;-(

So what's the problem?! What we want to be able to do is basically suspend the execution of the script at a certain point for either a specified (Delay) or an unspecified amount of time (#click). And we *do* have ways of doing this - our processes do *exactly* what we want to be able to do here.

So why not use processes?! Looking back at some older conversation about using multiple processes it appears to me that we've been holding back on it because of synchronization issues. While this is certainly understandable (we all know how hard process synchronization can be) I think that we can solve this problem in a very simple way that gives us the best of all worlds.

Here it is:

The basic problem that we have when using multiple processes is that we can't always say for sure when they are run. Because of this, the system may activate some process while we're in the midst of redrawing the screen, handling events, executing some (interfering) other process etc.

If not for the single problem of synchronization, using processes would solve many of the problems we're facing. Why?! Well, let's assume we would run those processes at a specific point in time (namely at the point where we invoke #step methods). If so, the system wouldn't be interrupted in an inconsistent state. Let's assume we would run the scripts in order (that is until they are either completed or suspended). If so, we would never have unexpectedly interfering process switches. And this means that have *exactly* the same semantics that scripts in eToys have (think about it: eToy scripts are run in order, in a certain time slot, until completion etc). Except that now we're carrying an execution environment with us, one that can be trivially suspended and resumed, one that can be reflected upon, one that's even efficient ;-)

So what if ... we just do that?! What if we define that "script process" and provide a default scheduling mechanism that avoids the more general problems?!

In my understanding, this would have some huge advantages such as:

* a "default" synchronization

Since script processes are scheduled in a specific time slot, the system is in a stable state. Since scripts are scheduled in order, they do not interfere with one another. In effect, we do provide a default synchronization mechanism which goes exactly as far as the eToy system today. This is neither to say that there will be no synchronization issues (you can write scripts in eToys suffering from lack of synchronization, but so can #step methods) nor that it's the ultimate solution for all users. All it does and all we need is to provide a useful default synchronization scheme.

* a clear separation of "script activation" and "message send"

In other words, invoking "self mumble" has the meaning of "run the script mumble in the current execution environment" whereas "self startScript: #mumble" means "schedule script mumble in a new execution environment".

* scripts can be "modal" and take time

It would be trivial to write any of the above examples the way they should be done. Since scripts carry the evaluation context we can just suspend them at the points we want to. This would allow us to have scripts that represent the equivalent to Wonderland scripts (which take time) while avoiding some of their complexities (no default sequencing).

[An interesting side effect here would be that a script that doesn't "take time" (e.g., does not invoke a method that implies delayed evaluation of sorts) could in effect be synchronous. So that "3 squared" = could effectively be equivalent to "3 startScript: #squared"].

* debugging scripts

One of the biggest problem we have is that eToy scripts are almost impossible to debug. If we encapsulate the user-script in its own process we could debug each script separately or stop execution of all scripts or whatever we think is reasonable. In short, we got a variety of options for the user. And, we could even being smart about the fact that we're debugging a user script and not a system process.

I do believe that tackling the inherent problem of synchronizing multiple processes in a way that (even if it wouldn't be considered straight-forward) can be explained to an omni-user will bring us a long, long way from where we are today. Since we could give those "user-scripts" the advantages of processes (carrying their own execution environment) as well as the advantages of the current "script scheduler" (ordered execution in a specific time slot) I don't see a reason not to go for multiple processes in user scripts.

Do you?

Cheers,

- Andreas