Subject: Re: NEW: Dispatcher (templateable dispatcher helper for MVC pattern and more)
Posted by mrjt on Thu, 15 Jul 2010 13:19:52 GMT
View Forum Message <> Reply to Message

I read the links you provided earlier. It sounds interesting, but I'm not sure I completely understood how it works.

My understanding:
Dispatcher/DispatcherQueue - These are threads and thread pools respectively, but differ from CoWork by not having static pools (so you can have several different pools per application).

Ports - A queue of typed data that can be linked to an Arbiter class. After registration items queued to the Port will (or more precisely depending on circumstances, 'may') prompt the Arbiter class to execute a threaded operation in a DispatcherQueue. Why they chose to name this thing 'Port' is beyond be, it's not a bad description but far to confusing. JobPort maybe, of DispatchPort would be better names.

Arbiter - This is where it gets a bit incomprehensible to me. To activate a Port you Call an Arbiter method detemine how it behaves (callback and activity), which creates a RecieverTask object that you then link with a DispatcherQueue object to active it. I don't understand why they've invented the Arbiter class when this stuff could just be done using the Port directly (one of the things that annoys me about .Net is the tendency to make things verbose and complex unnecessarily).

There are some other details that aren't mentioned, like if a Port has multiple registered RecieveTasks do they all get the data? I would assume so personally, but then what happens with the the queue if you have one that needs 10 strings and one that only wants 1? Some intelligence would be needed to deal with that one.

The 'Choice' design is the most interesting thing for me. I often find the problem with concurrency (in IO especially) isn't finding concurrent jobs to excute, but collecting their output in a sensible way later. I like the idea of executing a callback in a thread and also giving a set of callbacks to execute depending on the response of the first.

None of this should be to difficult to achieve in Upp, but I think your Dispatcher isn't quite the right approach. It will work very well as an event distribution system but for the CCR approach you really need to use Callbacks/Delegates.