Subject: Re: NEW: Dispatcher (templateable dispatcher helper for MVC pattern and more)
Posted by kohait00 on Thu, 15 Jul 2010 13:59:04 GMT
View Forum Message <> Reply to Message

Quote:
I think your Dispatcher isn't quite the right approach

no ofcorse, i know..it was just a start to play around but actually is definitly a different thing. (maybe to wet my own appetite

the whole story with arbiters and the like is indeed not that easy to grasp, it took me a while, and even now i need to think about it again and again to keep understanding .

based on your writing, you either downloaded and inspected CCR.Core.dll? the observations are pretty good

you are right about Ports. ports tend to have one Receiver normally, but not nessesaryly. they internally have a linked list for posted items and a linked list for attached Receivers (implementations of ReceiverTask) (both of same data type, but based on type free bases man thats complicated really).

arriving items are dispatched to the Receivers in round robin (AFAIK), but as you pointed out: to check conditions / setup a filter or so, there are the Predicates, which can be registered together with the handler for a ReceiverTask. they get the new item to see first. if they decide that it is to be used, the port dequeues the item, sets up a ITask (the user handler) and providing the item submits it to the DispatcherQueue (a task queue to the Dispatcher, a thred pool, scheduler, handling all registered DispatcherQueues in round robin). (actually the Receivers do the submitting to DispatchQueue work).

the really cool thing is that the arrived items are processed in parallel (depending on Receiver though, but normally). they wouldnt need a item queue either, if there were no such Receivers like Choice or Join which needs to store some items while the other involved ports dont meet the conditions.

the existing Receivers can even be nested. an interleave receiver (having concurrent, exclusinve and teardown [exit] receivergroups) can be composed of other Receivers like choice or joins.

its all about posting parallel tasks based on filtered income of objects. (even over serial channels, that makes it really funky). and even this is already supported in Upp with Stream / Serialize stuff).

the whole thing is pretty boosted indeed, but because it so flexible. i.e. we dont need Dispatcher that can handle multiple DispatcherQueues..

but it really sounds interesting. the design model is the dataflow, which is very logic. i'd like to give it a try, but its hard to go through the code and remember what was what.