Subject: NEW: Dispatcher (templateable dispatcher helper for MVC pattern and more)
Posted by kohait00 on Fri, 09 Jul 2010 11:24:23 GMT
View Forum Message <> Reply to Message

hi guys,

introducing here is the Dispatcher class, a templateable dispatcher with easy / simple / small interface (it a dispatcher, period).

```
template <class T>
class Dispatcher;
```

imagine cases where you need to pass object of information around, and have multiple destinations, maybe spread on various sheets of GUI, use Dispatcher to link them together and to forward stuff.

a class can have multiple

```
class MyCtrl
 : EditInt
 , public Dispatchable<Value>
 , public Dispatchable<int>
```

interfaces, needs to implement a

```
virtual void Dispatch(const T & o, unsigned param);
```

while param is just forwarded, Dispatcher does not care about it.

so the Dispatcher has got the same interface and just calls the same thing on each and every registered recepient.

it can use VectorMap internally, to help you access your stuff.

there is also a

```
class DispatcherGen;
```

a generic dispatcher, that can accept all types of Dispatchable<T> and dispatches only to supported instances.. it makes use of Any<> functionality, a genious class in Upp, just like One<> (not used here:)

the contained test applications demonstarete the use. there is not much to add as of documentations, it's a simple small class.

maybe you like it and add it to Bazaar.

cheers

File Attachments

1) Dispatcher.rar, downloaded 368 times

Subject: Re: NEW: Dispatcher (templateable dispatcher helper for MVC pattern and more)
Posted by kohait00 on Fri, 09 Jul 2010 11:25:56 GMT
View Forum Message <> Reply to Message

File Attachments

1) dispatcher.GIF, downloaded 977 times

Subject: Re: NEW: Dispatcher (templateable dispatcher helper for MVC pattern and more)
Posted by kohait00 on Mon, 12 Jul 2010 06:52:26 GMT
View Forum Message <> Reply to Message

some small bugfixes and changes:

the Dispatcher now is-a Vector, instead of have-a Vector. to let the programmer easily inspect stuff the way he wants.

GCC compileable, i missed to test compilation with gcc (mingw), so this one works.

i have changed the Dispatcher::Dispatch to Dispatcher::DoDispatch, so a Dispatcher could implement an own Dispatchable<T> interface , thus creating kind of Tree forwarder.

any suggestions welcome.

File Attachments

1) Dispatcher2.rar, downloaded 342 times

Subject: Re: NEW: Dispatcher (templateable dispatcher helper for MVC pattern and more)
Posted by koldo on Mon, 12 Jul 2010 07:02:59 GMT
View Forum Message <> Reply to Message

Hello kohait00

Sorry but I do not understand Dispatcher well.

Could you explain me as simple as possible (super dummy level) what it is for?

Subject: Re: NEW: Dispatcher (templateable dispatcher helper for MVC pattern and more)

Posted by kohait00 on Mon, 12 Jul 2010 07:51:59 GMT

View Forum Message <> Reply to Message

hey koldo..

it's a veeeery simple 1-to-many messages forwarder (dispatcher). the code doesnt do much more than

```
template<class T>
void Dispatcher<T>::DoDispatch(const T & o, unsigned param) const
{
 for(int i = 0; i < B::GetCount(); i++)
 {
  Dispatchable<T> * dest = B::operator[](i);
  if(dest)
   dest->Dispatch(o, param);
 }
}
```

where B:: is the base class container

the MVC pattern often times uses Dispatchers to pass around data, i.e. when a mode has many views, some genrated data needs to be displayed in each of the views, so it needs to be forwarded / dispatched.

imagine i.e your app generates data, that you want to be forwarded or displayed in more than one place. on multiple tabs or sth.. or some controller classes that listen for some kind of info to come in and generate more action.

it's actually a 'data link', 1-to-many.

for sure, the examples provided could be solved by just hooking up all the Ctrl's using callbacks. but this is too static, if zou imageine, you need to specify somehow dynamicly, who is about to receive stuff, or add or remove receivers, than your out. the hook up in code remains unchangable.

genrally, i think this Dispatcher thing also could be done using callbacks (a Vector<Callback>), but they tend not to enforce you to stick to some API, so using an interface class (Dispatchable<T>) makes sure you dont forget to implement the right function to receive stuff.

But i might try to provide a callback implementation as well, just for completeness.

i hope this was a bit more clear.

Subject: Re: NEW: Dispatcher (templateable dispatcher helper for MVC pattern and more)
Posted by koldo on Mon, 12 Jul 2010 09:37:01 GMT
View Forum Message <> Reply to Message

Thank you kohait00

I am beginning to understand.

Imagine I have an application with tabs, subtabs, windows, ...

Now the subtab "Calculations" generates data that has to be displayed in subtab "Graphics".

Until now I would done a global data repository and some functions to access it to put or get data.

Would Dispatcher class let to focus this in a better manner?

Subject: Re: NEW: Dispatcher (templateable dispatcher helper for MVC pattern and more)
Posted by kohait00 on Mon, 12 Jul 2010 11:43:38 GMT
View Forum Message <> Reply to Message

i dont know exactly how it could help here, the dispatcher is just a means to register multiple recipients for a typesafe messages.

what it could help is, ofcorse, to keep the data structure of ypur application clean. you can focus on data channels (Dispatchable<T>) and group things together, that belong together (recipients for the same data type)

i will try to provide some more examples on how to use the Dispatcher.

actually, the idea comes from one of my applications, where devices send data packets, that i want to be able to monitor in different ways, i.e. as a static text field, as an editfield somewhereelse, as a meter slider in some other place. so all the different controls simply listen for the data on their channels and display them, the dispatcher provides them with the data, and i only need to forward my stuff to the dispatcher, without need to take care everytime, that all controls get their info. thus the dispatcher was born. the version in my app is not that simple, the stuff i have learned there i tried to encorporate in this simple version.

basicly, the idea is not so new. dont know if you are familiar with Microsoft's DSS / CCR framework. they have the data flow control pattern (dont know if that oone exists, i am no prof). but the idea is that one bilds up the applications as a reactional framework. reacting on income of messages, an application can have multiple data slots, waiting for different types of messages,

having handlers that take care of each data element coming in. the total of that is, that you can create totally new applications by organizing and putting together other applications with their data channels linked together (even across networks, using serialisation of class instances). thats basicly the way a human beeing things of, when modeling a realworld problem. we tend to thing reactional not procedural.
anyway, to make a long story short: the dispatcher was kind of a step in this direction. as of now, ultimate has all the means to realize simiral behaviour. maybe we should go this direction as well to make Upp more attractive.

sorry for long post, sometimes its flowing

Subject: Re: NEW: Dispatcher (templateable dispatcher helper for MVC pattern and more)
Posted by mrjt on Mon, 12 Jul 2010 16:19:01 GMT
View Forum Message <> Reply to Message

Interesting. I have of course used a similar approaches but since it's never much work to implement a simple case I've never bothered with a generalised version.

I've looked at the code and have the following points/questions:
- What is the benefit of the optional the Ptr stuff? As far as I can tell the intention is that you must either a) unregister instances before destruction; or b) have PTEABLE enabled.

However, this code doesn't work in case (a):
```
 for(int i = 0; i < B::GetCount(); i++)
 {
                // #define PTEABLE code removed for clarity
  Dispatchable<T> * dest = B::operator[](i);
  if(dest)
   dest->Dispatch(o, param);
 }
```
because dest won't have been set to NULL if the class has been destroyed but not unregistered (as it would have if it was a Ptr).

- Since you are already storing a pointer to the Dispatcher in the Dispatchable, why don't you just Unregister the Dispatchable object in it's destructor. This would remove the need for the Dispatch object to be Pte derived to begin with, clean up your code and remove the need for client code to worry about unregistering except in special cases. Dispatcher would of course need to remain in a Ptr.

- Doesn't storing a pointer to the Dispatcher mean that an object can only be registered to a single dispatcher of a given type? It would be nice if a solution to that could be found.

- I also wouldn't inherit Dispatcher from Vector in the way you have. I can see why you might want to do it to expose the containers interface, but in this case you explicitly DONT'T want to do that.

- DispatcherGen is a very neat idea. I like your implementation with Any<> - Very clever

I got distracted (while I should be doing something less interesting!) and started fiddling with your code to incorporate my ideas above. I've attached my copy in case you're interested.

File Attachments

1) Dispatcher.zip, downloaded 592 times

Subject: Re: NEW: Dispatcher (templateable dispatcher helper for MVC pattern and more)
Posted by kohait00 on Mon, 12 Jul 2010 17:05:12 GMT
View Forum Message <> Reply to Message

Quote:
this code doesn't work in case (a):

you are absolutely right, i didnt bother too much about it. it's just been ideas that i made switchable, whichever later seems best, to remain.

Quote:
why don't you just Unregister in destructor

yes, thats the best way. in my first applications (from which the idea derived) i am actually doing this. i didnt want to make the first shot look too complicated, while the idea too simple i will add this thing ofcorse.

Quote:
I like your implementation with Any<> - Very clever

Any is clever, my stuff just uses it. i looked at Any, and 'loved' it, the static cache aproach with template function is just perfectly fast..

Quote:
but in this case you explicitly DONT'T want to do that.

why shouldnt i? the containers are a rational base. i imagine the precautional unregistering would be avoided when maipultating derectly the containers... but i had a hidden container version as well (see first version). so its basicly not decided. just need to know your point.

Subject: Re: NEW: Dispatcher (templateable dispatcher helper for MVC pattern and more)
Posted by kohait00 on Mon, 12 Jul 2010 20:56:04 GMT

View Forum Message <> Reply to Message

to show you some kind of far goal, just google for
'concurency and coordination runtime CCR'

http://en.wikipedia.org/wiki/Concurrency_and_Coordination_Ru ntime

i think ultimate brings in a lot about that, where Callbacks are a key part, CoWork is the thread pool, a Ports API should be designed and we got a slim CCR like aproach to creating application services (aka DSS, decentralized software services, another project that is directly linked to CCR). ultimate can make it that far, i know..

a brief overview, very intresting
http://msdn.microsoft.com/en-us/magazine/cc163556.aspx

PS: i have been dealing with M$ Robotics Studio and the DSS/CCR framework in C# and I loved it..because of the logical mesh up of services and applications. thats why i am thinking on how to immitate this in upp...

Subject: Re: NEW: Dispatcher (templateable dispatcher helper for MVC pattern and more)
Posted by kohait00 on Wed, 14 Jul 2010 14:23:02 GMT
View Forum Message <> Reply to Message

hey mrjt,

you wrote that i shouldnt expose container as class interface..
why?

Subject: Re: NEW: Dispatcher (templateable dispatcher helper for MVC pattern and more)
Posted by mrjt on Wed, 14 Jul 2010 14:38:29 GMT
View Forum Message <> Reply to Message

Because as you say, manipulation of the container without proper regard to registering/unregistering will lead to segmentation faults. One of the most important aspects of encapsulation and OOP is the avoidance of such issues. There is also a particular danger with regards to picking with Upp containers.

Besides that there isn't even much benefit. What are the chances of anyone ever needing anything besides add/remove?

Subject: Re: NEW: Dispatcher (templateable dispatcher helper for MVC pattern and more)

Posted by kohait00 on Wed, 14 Jul 2010 15:25:41 GMT
View Forum Message <> Reply to Message

ok, sounds good one probably never will have to access them directly, picking is a good point.

i will change stuff and provide you a version where i tried to incorporate your ideas with removing from containers as well.

Subject: Re: NEW: Dispatcher (templateable dispatcher helper for MVC pattern and more)
Posted by kohait00 on Wed, 14 Jul 2010 16:25:22 GMT
View Forum Message <> Reply to Message

here comes another version,

the key cache in Dispatchable<T> for multiple sources is kind of not sweet, but hard to avoid.

i did not implement the DispatcherGen as a container and forwarder for Dispatcher<T>, but might try it. nice idea.

File Attachments
1) Dispatcher3.rar, downloaded 325 times

Subject: Re: NEW: Dispatcher (templateable dispatcher helper for MVC pattern and more)
Posted by mrjt on Thu, 15 Jul 2010 10:24:20 GMT
View Forum Message <> Reply to Message

I'm struggling to understand why you're making it so complicated.

-If you are automatically unregistering Dispatchables on destruction you have no need to store Dispatchables in Ptr<>s and so can elliminate the overly complex PTEABLE branch.

-If you make DispatcherGen use Dispatcher objects internally (as I did) or even derive it from Dispatcher you can eliminate all of the complex Any stuff with sources. Keep it simple: Dispatcheables should only be registerable with Dispatchers and derived classes. Use the advantages of polymorphism.

-You don't need the source keys or the source key cache. If no key is specified by the caller to Unregister you can just iterate though the dests values looking for the correct pointer. This is potentially slightly slower, but if the developer is worried about performance then they should be using keys anyway.

-As I said before, do away with the NOMAP branch, the performance and memory usage gains are negligeable for the added code complexity. Having #defined branches through the code is really horrible, if you absolutely must have the branches then separate them completly or make them templated or something, currently it's extremely hard to read and understand the code.

Sorry, I don't want to seem too critical and my solution isn't the only way, but the whole thing just seems much more complicated (and thus diffcult to maintain and use) than it needs to be. Are you imagining some sort of use-case that I haven't considered?

Subject: Re: NEW: Dispatcher (templateable dispatcher helper for MVC pattern and more)
Posted by kohait00 on Thu, 15 Jul 2010 12:33:12 GMT

View Forum Message <> Reply to Message

Quote:elliminate the overly complex PTEABLE branch.
Ok, that should be no problem..

Quote:
make DispatcherGen use Dispatcher objects internally (as I did) or even derive it from Dispatcher

from which Dispatcher<T> should it derive? the goal is to have a generic Dispatcher, that can register arbitrary types. it wouldnt work becase DispatcherGen only would support one T again. the options is only to have a List / Vector / Array / Map of Dispatcher<T> inside to check then and forward to, or to use 'Container'<Any> to do it. i cant imagine that it would be that much of a big performance strike..look at the Any implementation, its a O(1) funtion call. but in huge amounts of different types, ofcorse yes.. so thats the 2 options.

Quote:
do away with the NOMAP , separate them completly

thats probably the best solution, dont worry, i hardly can read it myself i am not that much a fan of #ifdef branches

i will provide a clean package.

on more usecases: i am currently dot.net Reflector'ing the Microsoft.CCR.Core.dll to get a grisp on how they have done this whole Ports and Arbiters thing, believe it or not, 50% of that is already provided in Upp, named CoWork and Callbacks . this is actually the direction i want to go for..

dont know if you ever used Microsoft Robotics Development Studio (current RC3 is for free again). but the design patterns are really coool. so i'd like to have some of the benefits in upp as well.

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.

Subject: Re: NEW: Dispatcher (templateable dispatcher helper for MVC pattern and more)
Posted by mrjt on Thu, 15 Jul 2010 13:24:13 GMT
View Forum Message <> Reply to Message

Ignore, pressed the wrong button

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.

Subject: Re: NEW: Dispatcher (templateable dispatcher helper for MVC pattern and more)
Posted by mrjt on Thu, 15 Jul 2010 14:15:25 GMT
View Forum Message <> Reply to Message

Wow, It's even more complicated than I thought

I can see why this would be very useful for robotics. You can trigger a whole tree of asyncronous motorised actions from a single command and then link further actions to be executed once that completes. That's exactly what you need for moving a robot arm for instance.

Subject: Re: NEW: Dispatcher (templateable dispatcher helper for MVC pattern and more)
Posted by kohait00 on Thu, 15 Jul 2010 14:21:21 GMT
View Forum Message <> Reply to Message

it's even better, combined with their DSS runtime, such applications become services that provide some main ports outwards, and the runtime ensures that an orchistration (a set of independantly started services) get linked together (trough their data channels) a mesh-up application is born. a roboter can then, if available use a web cam service or a speech service, depending on provided / started hardware.. this is really cool stuff. unfortunately its M$

Subject: Re: NEW: Dispatcher (templateable dispatcher helper for MVC pattern and more)
Posted by kohait00 on Fri, 16 Jul 2010 07:37:24 GMT
View Forum Message <> Reply to Message

i am already stitching together some classes, i think by the end of next week we could have a first version to demonstrate at least simple receiver behavior..

but an important thing is the Callback extension provided in another post of mine. without that i wont be able to do the stuff. maybe you can spread the word on it

Subject: Re: NEW: Dispatcher (templateable dispatcher helper for MVC pattern and more)
Posted by kohait00 on Thu, 05 Aug 2010 08:19:15 GMT
View Forum Message <> Reply to Message

the overall discussion for CCR is moved to
http://www.ultimatepp.org/forum/index.php?t=msg&goto=276 19&#msg_27619

there is also available an implementation of the first CCR arbiters and receivers..

Subject: new stuff
Posted by kohait00 on Thu, 05 Aug 2010 11:39:30 GMT
View Forum Message <> Reply to Message

hi mrjt,

here comes another version, where i tried to simplify what was possible. and i added a DispatcherCB, which works with Callbacks, but ofcorse needs keys. TEstDistapcher6 is for it to be tested.

like you said, i removed the PTEABLE and NOMAP stuff. it's pretty small now ..

File Attachments

1) Dispatcher4.rar, downloaded 292 times