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 559 times