AndyPatterns

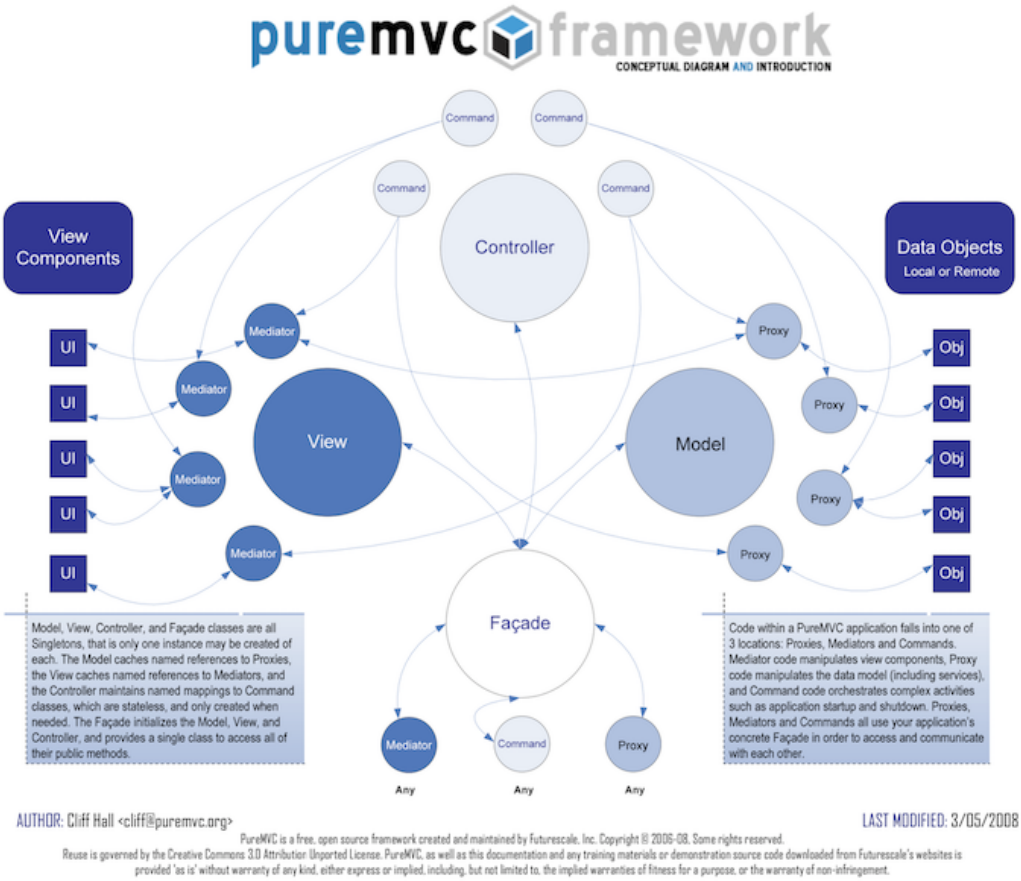
PureMVC - Minimal wxPython Example

Understanding PureMVC can be hard, even though it is a relatively simple framework. That's why a minimal example can help. Here is one I created for wxPython and Python users. This will run fine on windows, mac and linux. Install the PureMVC python port and wxPython (which is already present on Mac 10.5) and you should be able to run it.

Incidentally there is a helpful minimal example for Actionscript users too. To actually run that example in Flex you need to create a Flex project, import the source, move the startup function found in main.as into the mxml, and trigger it via a onloadcomplete event in the mxml. Leave the startup code exactly the way it is - passing in the stage as a parameter to the startup code is fine, even in Flex.

A tip on understanding PureMVC

A potential pitfall to understanding PureMVC is that you merrily visit the puremvc.org website and start to read about the wonderful architecture of the framework - the facade, the controller and its commands, the view with its mediators and the proxies with their references to your model. You read about what notifies what, who points where - then you look in awe at the wonderful framework diagram:



and wonder where to get started! The trouble is, you don't need to understand the intricate details of how the framework itself works in order to use it (though you should eventually read all the PureMVC documentation and even read the PureMVC source code - its a suprisingly small amount of code).

In fact all you need to create in order to *use* the framework is a concrete **facade** class in order to define your notification messages, a **mediator** class for any gui that you have, a **proxy** class for you model class and finally one ore more **command** classes depending on what behaviour you want to have.

Overview of the Python Minimal Example

This demo illustrates the simplest way (that I could think of) to build a PureMVC application in python using wxPython as the GUI. A string in the model is displayed in the GUI in a textfield. Anything you type gets converted to uppercase and stored back in the model, courtesy of a command class triggered when you hit the ENTER key in the GUI.

We are modelling and visualising a single string of a model class Data. If you want to imagine a sequence diagram... On startup, the model's data is pushed to the GUI. When the user hits ENTER in the GUI, the mediator picks up on this wx event and broadcasts the pureMVC message DATA_SUBMITTED. A command class thus gets triggered which converts the text to uppercase (this is the business logic) and stuffs the info into the model. The model's setter broadcasts a DATA_CHANGED message, which the mediator intercepts, updating the GUI.

Thus anything you type gets converted to uppercase and stored in the model.

Code

The single python file can be downloaded from here.

```
import wx
import puremvc.interfaces
import puremvc.patterns.facade
import puremvc.patterns.command
import puremvc.patterns.mediator
import puremvc.patterns.proxy


"View"
MyForm is the GUI, built in wxpython. MyFormMediator is the mediator which knows
about and specifically looks after MyForm, intercepting wx events from the GUI
and broadcasting pureMVC notification messages. The mediator also listens for
pureMVC notification messages and stuffs data back into the GUI. The "View" in
pureMVC is the entire system comprised of class View (part of the pureMVC
framework) which holds references to one or more mediators, each of which looks
after a GUI form or part of a GUI.

class MyForm(wx.Panel):

 def __init__(self, parent):
 wx.Panel.__init__(self,parent,id=3)
 self.inputFieldTxt = wx.TextCtrl(self, -1, size=(170,-1), pos=(5, 10), style=wx.TE_PROCESS_ENTER)

class MyFormMediator(puremvc.patterns.mediator.Mediator, puremvc.interfaces.IMediator):
 NAME = 'MyFormMediator'

 def __init__(self, viewComponent):
 super(MyFormMediator, self).__init__(MyFormMediator.NAME, viewComponent)
 self.viewComponent.Bind(wx.EVT_TEXT_ENTER, self.onSubmit, self.viewComponent.inputFieldTxt)

 def listNotificationInterests(self):
 return [ AppFacade.DATA_CHANGED ]

 def handleNotification(self, notification):
 if notification.getName() == AppFacade.DATA_CHANGED:
 print "handleNotification (mediator) got", notification.getBody()
 mydata = notification.getBody()
 self.viewComponent.inputFieldTxt.SetValue(mydata)

 def onSubmit(self, evt):
 mydata = self.viewComponent.inputFieldTxt.GetValue()
 self.sendNotification(AppFacade.DATA_SUBMITTED, mydata)


"Controller"
StartupCommand and DataSubmittedCommand are the command classes, triggered by
notification messages AppFacade.STARTUP and AppFacade.DATA_SUBMITTED
respectively. The "Controller" in pureMVC is the entire system comprised of
class Controller (part of the pureMVC framework) which holds references to one
or more commands.

class StartupCommand(puremvc.patterns.command.SimpleCommand, puremvc.interfaces.ICommand):
 def execute(self, notification):
```

```
 print "startup execute (command)"
 wxapp = notification.getBody()
 self.facade.registerMediator(MyFormMediator(wxapp.myForm))
 self.facade.registerProxy(DataModelProxy())

class DataSubmittedCommand(puremvc.patterns.command.SimpleCommand, puremvc.interfaces.ICommand):
 def execute(self, notification):
 print "submit execute (command)", notification.getBody()
 mydata = notification.getBody()
 self.datamodelProxy = self.facade.retrieveProxy(DataModelProxy.NAME)
 self.datamodelProxy.setData(mydata.upper())
```

"Model"

DataModelProxy is the proxy for class Data. The "Model" in pureMVC is the entire system comprised of class Model (part of the pureMVC framework) which holds references to one or more proxies (e.g. DataModelProxy) each of which looks after a model (e.g. Data). Note that a model class could be more complex and the associated proxy could look after looping and pulling out larger chunks of information.

```
class DataModelProxy(puremvc.patterns.proxy.Proxy):
 NAME = "DataModelProxy"

 def __init__(self):
 super(DataModelProxy, self).__init__(DataModelProxy.NAME, [])
 self.realdata = Data()
 self.sendNotification(AppFacade.DATA_CHANGED, self.realdata.data)

 def setData(self, data):
 self.realdata.data = data
 print "setData (model) to", data
 self.sendNotification(AppFacade.DATA_CHANGED, self.realdata.data)

class Data:
 def __init__(self):
 self.data = "Hello - hit enter"
```

"Facade"

The facade is a singleton and is where you define all messages types (all just strings) and where you associate command classes with particular messages. There can be messages NOT associated with commands, which are used for mediators to listen to (e.g. model indirectly talking to mediators).

```
class AppFacade(puremvc.patterns.facade.Facade):
 STARTUP = "STARTUP"
 DATA_SUBMITTED = "DATA_SUBMITTED"
 DATA_CHANGED = "DATA_CHANGED"

 @staticmethod
 def getInstance():
 return AppFacade()

 def initializeController(self):
 super(AppFacade, self).initializeController()

 super(AppFacade, self).registerCommand(AppFacade.STARTUP, StartupCommand)
 super(AppFacade, self).registerCommand(AppFacade.DATA_SUBMITTED, DataSubmittedCommand)

 def startup(self, app):
 self.sendNotification(AppFacade.STARTUP, app);

class AppFrame(wx.Frame):
 myForm = None
 mvcfacade = None

 def __init__(self):
 wx.Frame.__init__(self,parent=None, id=-1, title="PureMVC Minimalist Demo",size=(200,100))
 self.myForm = MyForm(parent=self)
 self.mvcfacade = AppFacade.getInstance()
 self.mvcfacade.startup(self)

class WxApp(wx.App):
 appFrame = None

 def OnInit(self):
 self.appFrame = AppFrame()
 self.appFrame.Show()
 return True
```

```
if __name__ == '__main__':
 wxApp = WxApp(redirect=False)
 wxApp.MainLoop()
```

Notes

Pardon the pun but PureMVC often uses 'note' to refer to the notification message parameter, which I found confusing. For example when a meditor receives a notification. I use **notification** as the parameter name.

Notification messages have a .getName(), .getBody() and .getType(). Except for the name, you can pass any info you like in the second two parameters, its up to you. Typical parameters are references to forms, apps, data that has changed, reference to objects, other string messages of your own devising. You could use more than one parameter if say, you wanted to broadcast both the data that changed and an actual reference to the object whose data had changed.

Alternative implementation - even simpler!

Here is an even simpler implementation, essentially the same, however I have removed the startup command. You can initialise the bits and pieces of the application, creating the mediator and model proxy etc. in regular code rather than going through all the fancy startup command stuff (see the bold code). This helps reduce complexity and increases your chances of understanding what is going on with the PureMVC architectural pattern. In practice, in a complex project, a startup command may very well be a good idea.

You can download this example here.

```
import wx
import puremvc.interfaces
import puremvc.patterns.facade
import puremvc.patterns.command
import puremvc.patterns.mediator
import puremvc.patterns.proxy

class MyForm(wx.Panel):

 def __init__(self, parent):
 wx.Panel.__init__(self,parent,id=3)
 self.inputFieldTxt = wx.TextCtrl(self, -1, size=(170,-1), pos=(5, 10), style=wx.TE_PROCESS_ENTER)

class MyFormMediator(puremvc.patterns.mediator.Mediator, puremvc.interfaces.IMediator):
 NAME = 'MyFormMediator'

 def __init__(self, viewComponent):
 super(MyFormMediator, self).__init__(MyFormMediator.NAME, viewComponent)
 self.viewComponent.Bind(wx.EVT_TEXT_ENTER, self.onSubmit, self.viewComponent.inputFieldTxt)

 def listNotificationInterests(self):
 return [ AppFacade.DATA_CHANGED ]

 def handleNotification(self, notification):
 if notification.getName() == AppFacade.DATA_CHANGED:
 print "handleNotification (mediator) got", notification.getBody()
 mydata = notification.getBody()
 self.viewComponent.inputFieldTxt.SetValue(mydata)

 def onSubmit(self, evt):
 mydata = self.viewComponent.inputFieldTxt.GetValue()
 self.sendNotification(AppFacade.DATA_SUBMITTED, mydata)

class DataSubmittedCommand(puremvc.patterns.command.SimpleCommand, puremvc.interfaces.ICommand):
 def execute(self, notification):
 print "submit execute (command)", notification.getBody()
 mydata = notification.getBody()
 self.datamodelProxy = self.facade.retrieveProxy(DataModelProxy.NAME)
 self.datamodelProxy.setData(mydata.upper())

class DataModelProxy(puremvc.patterns.proxy.Proxy):
 NAME = "DataModelProxy"

 def __init__(self):
 super(DataModelProxy, self).__init__(DataModelProxy.NAME, [])
 self.realdata = Data()
 self.sendNotification(AppFacade.DATA_CHANGED, self.realdata.data)

 def setData(self, data):
 self.realdata.data = data
 print "setData (model) to", data
```

```
  self.sendNotification(AppFacade.DATA_CHANGED, self.realdata.data)

class Data:
 def __init__(self):
 self.data = "Hello - hit enter"

class AppFacade(puremvc.patterns.facade.Facade):
 DATA_SUBMITTED = "DATA_SUBMITTED"
 DATA_CHANGED = "DATA_CHANGED"

 @staticmethod
 def getInstance():
 return AppFacade()

 def initializeController(self):
 super(AppFacade, self).initializeController()
 super(AppFacade, self).registerCommand(AppFacade.DATA_SUBMITTED, DataSubmittedCommand)

class AppFrame(wx.Frame):
 myForm = None
 mvcfacade = None

 def __init__(self):
 wx.Frame.__init__(self,parent=None, id=-1, title="PureMVC Minimalist Demo",size=(200,100))
 self.myForm = MyForm(parent=self)
 self.mvcfacade = AppFacade.getInstance()
 self.mvcfacade.registerMediator(MyFormMediator(self.myForm))
 self.mvcfacade.registerProxy(DataModelProxy())

class WxApp(wx.App):
 appFrame = None

 def OnInit(self):
 self.appFrame = AppFrame()
 self.appFrame.Show()
 return True

if __name__ == '__main__':
 wxApp = WxApp(redirect=False)
 wxApp.MainLoop()
```

Conclusion

I hope this minimalist example helps you understand PureMVC. Of course read the pdf documentation on PureMVC by Cliff. For a step by step guide to buildng a PureMVC based application in wxPython - or any other language for that matter, see my blog entry Refactoring to PureMVC.

-Andy

Comments:

Posted by Toby de Havilland on Mar 4th, 2009
Excellent example Andy, great stuff.

Posted by diamondtearz on Dec 10th, 2009
Wow! Thanks for putting this out there. I've recently jumped on the Python boat and have used PureMVC in Actionscript. I was looking for a good example to get me going and this fits the bill perfectly.

Posted by Fabio on Jan 18th, 2014
Dear Andy,
I hope you're still around after all this time. Sorry but I found your wonderful blog just now! Your example is very nice, but, unfortunately, I don't get it to work. There is a very strange Traceback here that I cannot really get rid of. I'm running python 2.7.3 on Windows (8...)
I cannot really understand why this strange error. self.realdata is absolutely well defined!

```
C:\NewPython27\Scripts\python.exe C:/Users/Fabio/Documents/python/puremvcminimalwx0.py/fromSite.py
startup execute (command)
Traceback (most recent call last):
File "C:/Users/Fabio/Documents/python/puremvcminimalwx0.py/fromSite.py", line 156, in
wxApp = WxApp(redirect=False)
File "C:\NewPython27\lib\site-packages\wx\_core.py", line 7978, in __init__
self._BootstrapApp()
File "C:\NewPython27\lib\site-packages\wx\_core.py", line 7552, in _BootstrapApp
return _core_.PyApp__BootstrapApp(*args, **kwargs)
```

```
File "C:/Users/Fabio/Documents/python/puremvcminimalwx0.py/fromSite.py", line 151, in OnInit
self.appFrame = AppFrame()
File "C:/Users/Fabio/Documents/python/puremvcminimalwx0.py/fromSite.py", line 145, in __init__
self.mvcfacade.startup(self)
File "C:/Users/Fabio/Documents/python/puremvcminimalwx0.py/fromSite.py", line 135, in startup
self.sendNotification(AppFacade.STARTUP, app);
File "C:\NewPython27\lib\site-packages\puremvc\patterns\facade.py", line 265, in sendNotification
notificationName, body, noteType
File "C:\NewPython27\lib\site-packages\puremvc\patterns\facade.py", line 284, in notifyObservers
self.view.notifyObservers(notification)
File "C:\NewPython27\lib\site-packages\puremvc\core.py", line 306, in notifyObservers
obsvr.notifyObserver(notification)
File "C:\NewPython27\lib\site-packages\puremvc\patterns\observer.py", line 88, in notifyObserver
self.getNotifyMethod()(notification)
File "C:\NewPython27\lib\site-packages\puremvc\core.py", line 91, in executeCommand
commandInstance.execute(note)
File "C:/Users/Fabio/Documents/python/puremvcminimalwx0.py/fromSite.py", line 93, in execute
self.facade.registerProxy(DataModelProxy())
File "C:/Users/Fabio/Documents/python/puremvcminimalwx0.py/fromSite.py", line 106, in __init__
super(DataModelProxy, self).__init__(DataModelProxy.NAME, [])
File "C:\NewPython27\lib\site-packages\puremvc\patterns\proxy.py", line 46, in __init__
self.setData(data)
File "C:/Users/Fabio/Documents/python/puremvcminimalwx0.py/fromSite.py", line 111, in setData
self.realdata.data = data
AttributeError: 'DataModelProxy' object has no attribute 'realdata'
```