Subject: MVC example
Posted by rochk on Mon, 17 Mar 2008 22:32:51 GMT
View Forum Message <> Reply to Message

Hi there!

I have just started experimenting with U++. Mainly playing around with the samples and tutorials included in the installation package. I find it quite educational - but I miss one thing: real MVC (Model-View-Controller) paradigm example. I am planning to create quite complicated application and I would like to start with the good code design. Anyone knows where to find U++ MVC example? Anyone can create a simple example for this?

BR
/RochK

Subject: Re: MVC example
Posted by Mindtraveller on Tue, 18 Mar 2008 10:17:42 GMT
View Forum Message <> Reply to Message

In my opinion U++ is standing aside of MVC paradigm.
First of all, MVC assumes a number of child windows inside one parent window (MDI). As far as I know, U++ doesn`t support this paradigm. Yes, you may think it`s that bad. But in my opinion U++ supports much more flexible (and easy to use!) paradigm.

U++ has innovatively attractive support for multi-tab and even multi-window design approach. Of course you have your class which describes one unit of what you edit or view. This can be file/item/etc. U++ approach "everything belongs somewhere" assumes that your class has all requiring GUI objects inside of it. This is exactly what I expect: having GUI controls inside my class as private members nicely fits into OOP paradigm and is what I consider an obligatory part of *good class design*.

These GUI objects are children of Layout class object, which is working not only as simple container, but as layout for them too. Of course, this layout is edited visually in theIDE at design time.

So the general idea is: create layout file with theIDE designer. Include corresponding layout you made as class to your class and have fun. Each time you create your class object, GUI is automatically created and attached where you need it (new tab or new window, or even new docking control).

Let`s take a real example of multi-tabbing interface. Look at simplified version of class COManager. It represents some multi-threading server, working in background. Data it processes is drawn inside it`s tab layout. All we have to do is add parentCtrl member into my class and have it initialized in the constructor.

```
// .h
#define LAYOUTFILE "COManager.lay" //contains InputLayout
```

```
#include <CtrlCore/lay.h>

class COManager
{       typedef COManager CLASSNAME;

public:
 COManager(TabCtrl &tabs, int comN);
 ~COManager();
private:
 WithInputLayout<ParentCtrl> parentCtrl; //generated by theIDE in .lay file
};

// .cpp
COManager::COManager(TabCtrl &tabs, int _comN)
{
 CtrlLayout(parentCtrl); //init layout
 parentCtrl.SizePos(); //make it sizeable
 tabItem = & tabs.Add("...").Slave(&parentCtrl); //add new tab to tab control
}
```

That`s it! It is that simple.
If after a month of usage I decide to have multiple windows insted of multiple tabs inside one window (which is mostly bad decision) all I change is "tabs.Add" to a single line creating new window with my layout. I think that it is exactly what is needed for well structured and rapid GUI development.

Subject: Re: MVC example
Posted by Mindtraveller on Tue, 18 Mar 2008 13:05:53 GMT
View Forum Message <> Reply to Message

More of that.
I think that MVC paradigm is some kind of glue attaching bad program design to OOP. According to OOP paradigm, each class must have everything possible within it. It must be it`s own model, view and controller - in one place. Or you will have too strong dependencies between different parts of program. This makes it hard to debug and a complete nightmare to upgrade.

My experience acknowledges this approach - less others know about your class internal structure - better your life is. In Borland C++ Builder I had a nightmare of sending UI events between classes hierarchy levels. This all was because of the fact that GUI objects could belong only to main window class. But this is wrong approach: most of time this window has nothing in common with those controls which serviced some internal objects. And I had very long series of useless member function calls - everything in attempt to comply to OOP paradigm. It was horrible but a bit better than classic MVC idea:

```
//MyMainWindow.cpp
void MyMainWindow::OnThisButtonClick (TObject *Sender)
{
   workManager::ActivateSomething();
```

```
}

//WorkManager.cpp
void WorkManager::ActivateSomething()
{
   if (currentTab < 0)
      return;
   workersArray.GetItem(currentTab)->DoSomething();
}

//Worker.cpp
void Worker::DoSomething()
{
   ParallelWorker::StartTask();
}

//ParallelWorker.cpp
void ParallelWorker::StartTask()
{
   //DOING ACTUAL WORK
}
```

And all this code is because of unnatural default position of controls inside window/form class. Dynamic creating/deleting of controls was i nightmare too because it is hard to synchronize creating of Workers with it`s controls.

The situation described is far *better* than classic MVC approach, where all the data is public.

So U++ approach in my opinion meets global OOP requirement having everything about this class inside this class. Now I simply write:

```
void ParallelWorker::ParallelWorker(SomeKindOfParent &parentContainer)
{
   CtrlLayout(myLayout);
   parentContainer.AddUnit(myLayout);
   myLayout.thisButton <<= THISBACK(StartTask);
}

void ParallelWorker::StartTask()
{
   //DOING ACTUAL WORK
}
```

It is a way shorter, 10 times as more readable, and more important - inside my class. Now I`m free of debug & upgrade nightmare.

Subject: Re: MVC example
Posted by rochk on Tue, 18 Mar 2008 19:08:42 GMT
View Forum Message <> Reply to Message

Dear Pavel,

Thank you for sharing your point of view on that matter. However I have to say it sounds quite strange to me when I recall all the books saying that Model should have nothing in common with View. I am creating my applications starting from PDC (Problem Domain Components - in other words: Model) and testing it when it is ready to do something useful in CLI (Command Line Interface). After this I am creating HIC (Human Interface Components - in other words: View). That way leads to a growing collection of higly specialized classes which can be reused in new projects. Including some GUI members in Model classes is something that I can not understand and even more - don't wanna do.

So, anyone can show me simple but complete U++ application built according to MVC paradigm? As a reference of course...
To encourage you a little bit to help me - attached you will find very simple U++ project with working model classes - it uses STL container and string classes - because I am still not sure which GUI library I will use in my project. All I ask you is to show me how to design View and Controller in U++. The model classes are: Client, Transaction, Line_Item and Product. The application creates demo Customer with some Transactions connected. I would like to have a simple dialog displaying Customer name and addres, his Transaction list and another list with Line_Items reflecting currently selected Transaction. That's it...

BR,
/RochK

File Attachments
1) Rachunki.7z, downloaded 667 times

Subject: Re: MVC example
Posted by zsolt on Tue, 18 Mar 2008 22:04:38 GMT
View Forum Message <> Reply to Message

In ancient times there were no useful gui toolkits nor simple database interfaces, so creating gui and storing/retrieving data were very complex tasks.
So some clever programmers started separating these ugly code from the main logic of the program.
The naming of these separated parts were View and Model and they labeled the main logic of the program as Controller.

In U++, Model and View are so simple, that some people think, that they are not exist at all.
But View is simply in the .lay file, Model in the .sch file and Controller is your C++ code.

Of course, model can be a single file too and you can use the extremely simple serialization API in U++ for simple file storage.

The problem is, that on some universities, professors are sticked in the past, and can not understand, that MVC is a workaround to a currently not existing problem.

Edit: check SQLApp or HomeBudget in examples as simple examples.

Subject: Re: MVC example
Posted by unodgs on Tue, 18 Mar 2008 22:48:55 GMT
View Forum Message <> Reply to Message

Well said zsolt

Subject: Re: MVC example
Posted by cbpporter on Wed, 19 Mar 2008 06:58:02 GMT
View Forum Message <> Reply to Message

If you want pure MVC, IMO U++ is not best suited for you. For one, a lot of GUI components which represent collections own their own data, so basically they are both views and models in the same time. You may think of this as unnatural from a MVC perspective, but in practice it makes a lot of sense for the control both to own and manage it's data. Also, MCV imposes some tight coupling between fairly redundant classes. While that coupling is tight logically, it it pretty loose from structural point of view (which means you have to use glue code to to tie components together), which does not fit that well with U++ approach, where everything belongs somewhere, rule imposed also by code layout.

But if U++ was not built for MVC, I could accomplish it. It has something called a Display, which is a view used to alter the visual representation of the data from a control. Look over the reference/Display example to see a very simple example of it. Here you have a DropList which owns a list of fonts, providing default view for it's data, and that view is replaced with a custom one, which this time is a pure view. Such uses of explicit views is pretty much all you'll ever need in practice. The list box could have contained any data, which means you don't have to embed GUI items in your model.

Other than this, I can't give you any more samples of MVC pattern from U++ because I'm not aware of any. But, I'm also not aware of any success stories of applications written from this perspective. There is a huuuuge commercial MFC application which uses an imposed MVC model (not a pure one), and I can't stop cursing it because of it's unnecessary casts and other awkwardnesses that it imposes.

Subject: Re: MVC example
Posted by mirek on Mon, 24 Mar 2008 11:00:20 GMT
View Forum Message <> Reply to Message

Well, my take is that MVC is not a good abstraction for many real-world problems.

It has a good sense only if you bind more views to single data model. And yes, in U++, you would need a little bit more of code to achieve this (or, OTOH, you can fake it in most cases too, just look at how TheIde does text window split .

Sometimes, it also seems to me that rigid MVC is not that flexible. Well, my knowledge of MVC perhaps does not go too deep, but IMO, good example is RichText as "model" and RichEdit as "view/controller". But AFAIK, for pure MVC, RichText should not have the means of painting itself, right? But that would make the code 200% more complicated...

Mirek

Subject: Re: MVC example
Posted by kohait00 on Thu, 04 Aug 2011 22:29:12 GMT
View Forum Message <> Reply to Message

MVC in upp is actually a very interesting topic to talk about, because it leads to proper structurizing of your app very quick. since i'm currently dealing with some related stuff in my project, i've been messing around a bit and will rewarm the thread i dont plan to introduce MVC in upp, but to see and maybe show some hints on how/why upp is still able to be used with MVC patterns to a certain degree, though it is not quite easy. and we can't deny that sometimes kind-a-MVC is at least a bit appealing.

first things first: everything dependands on *what* you define to be your model. then, it points you what you define to be the view and last, what logic to lay as base (control).

if you take your whole upp application base to be the model (SQL backend, some sort of state data, which doesnt know anything of it beeing used or edited, etc) then it will logically draw you to the point to see the upp layout of the app as the view, beeing the Ctrl's the processors of user interaction (forwarded to the control to perform the logic on the model). MVC states, that a view is exchangeable, which is kind a irrelevant here, since it's your single application instance, except for if you want your app be a chameleon and *totaly* change its appearance. additionally, you normally got one single control in this case, which is, according to MVC pattern, your application logic, your current handling context logic. so far, upp quite well complies to MVC, except for one thing: the general notification mechanism from the model, which instructs one or multiple views to perform a refresh of the whole representation. in case of an upp application, this is sort of done with arbitrary means, and no more than one observer (the app view) is needed to be listening. so somehow, the view is triggered to perform a redraw. that's fine.

things significantly change, when you reduce your model, or split your app to be comprised of several models, with it's independant views. now the lack of notification means becomes evident, since more than one view could be rendering the same model and need to keep track of changes.

let's look at Display: it's not a true view according to MVC. it should have means of receive and forward userinput in some fashion to the controller. Display is view-only.
also, Display is rendering Value as 'Model', which lacks notification mechanism, though it refers to

the same 'Data' internally, beeing sort of a shared ptr. a change to the data does not trigger all pointing-to instances of Value to signal a Display to redraw itself.. and how should 'control' come into play here

this quickly leads to looking at the Upp Ctrl's, since they are MVC all in one, just as koldo pointed out. the draw mechanism is the view, also processing the user input, which is used in a cotnrol mechanism internally. and a upp Ctrl renders and controls an internal form of Value as model, very generally speking. it can be accessesed with GetData/SetData. noone else acutally 'views' the same internal model of the Ctrl. the Ctrl can be notified of its internal model's change (after manually calling SetData, i.e.) from outside and have it redraw itself, by calling Refresh() (or UpdateRefresh() if more internal calculations are needed). on the other hand, an Upp Ctrl offers means of notification of change of the interlan model due to user handling. CtrlCore::WhenAction is a settable Callback, called each time some user input causes a change of the internal model. so the Ctrl itself, not the model, is notifying. it can be used to notify others based on that model. it wont notify you of changes made through code api though (after SetData directly, i.e). you need to call Action() if you want so. but it will save you from a lot of loop call headache. see Ctrl design aspects in srcdoc of CtrlCore. we see: Ctrls are a sort of MVC in this sense: single-internal-M + single-parametrizable-V + single-internal-C.

now, using Ctrls as MVC has caveats. each Ctrl can store its own model (Value), and it notifies you of its own models' changes only. and the internal model is not neccessarily a Value, it can merely be set/read using Value, but it might map to other things internally, totally unrelated to Value that just happen to be exposed through the GetData/SetData gateway (but not neccessarily only through it). so best way is, to see upp::Ctrl's VC only, since they render a meta-model..which is stored internally and not fully known to anyone else, who also maybe wants render it.
also, the notification mechanism is very uncomfortable here. Callback usually only has a one-destination-call, so only one observer could be notified. one can add more observers with 'WhenAction << THISBACK(notifymecb);', but in therms of subscription handling, this is totally unsatisfactory. how to unsubscribe now? WhenAction.Clear() would flush other recepients as well. so definitely no notification mechanism here, suited for MVC.

IF using Value as the model of choice, which is a good idea, since it is a common data base in upp for all sorts of data, then one needs to have a change of perspective here. see the Ctrl internal Value as 'copy' of your model, which the Ctrl needs to have to be able to show something. the real model lies somewhere outside and needs to provide for the notification mechanism. now, the Ctrl becomes a View with means of receiving user input, which needs to have mapped its actions to some sort of control/logic, which in turn would cause the model change and would notify all other managed views (Ctrls) of that model of the changes and cause their redraw.

i came up with the Dispatcher package in bazaar, exactly due to such a problem. use case: an audio controlling software, operating on a database of objects (each object is a seperate model), which represent some parameters of an audio device (gain, mute, etc). each object can be represented in various ways, i.e mute object can be shown as a LED (readonly V, C is empty) on the main view of the device page, and have a button set on it, which can control it's state. (here the button is VC). so the need of a round-robin notification mechanism becomes evident. and upp does not provide for theese use cases.

but it provides a lot of very interesting helpers to accomplish even such a task, i.e. Value class,

Callback class..doing some extra work, some extra template mapping classes lets you have such a use case completed. i am currently reworking this thing to be more general. so maybe it can be used by others in some way.

generally speaking, mirek is right. MVC is to unspecific for real world problems. that's why it's a pattern and not a library . the pattern applys well if the conditions are met (the need of a central model with many views). but the solution for this is oftentimes tied to the things you have to deal with anyway, API, class base, etc. so a general answer or even implementation is dificult here anyway. nevertheless, i will share the object implementation soon. maybe it's general enough for the crowd.. the base ide is this:

object is a Value based model, with a Dispatcher to notify observers. observers can be other controls (upp controls with extended capabilities) or some actors (mappers to actions or callbacks). an application, that has a portion of MVC-like problems, can use a dictionary of objects and set up some VC parts on it, and wire up some actions to the objects. it's still not 100% MVC, but it's relatively general and flexible. this also enables to have a live work environment where one can select / create / move arbitrary Ctrl's and wire them with some objects to monitor and edit parameters/actions. the application internally can pre-wire some action to the objects' cataloge.

with this long post i hope to have condensed some thoughts on MVC and UPP, since it might be an ever regressing question. maybe some parts of the stuff discussed in this thread could go to 'wetting your appetite' or general questions.

sidenote:

android is MVC as well. the model is even stored in the cloud . one can obtain some providers to access the information needed (not all apps really do need such stuff). the control is carried out in the app itself, using the android controls as views..the dispatcher/notifier here is google itself

---

Subject: Re: MVC example
Posted by kohait00 on Fri, 05 Aug 2011 08:41:40 GMT
View Forum Message <> Reply to Message

mirek pointed out that the ide uses a certain mechanism to i.e. have a Split view of probably the same document.
i've reviewed the stuff and it turns out that the solution here is quite simple.
ide.cpp:795

```
void Ide::Periodic()
{
 CheckFileUpdate();
 ...
}
```

ide.win:621

SetTimeCallback(-20, THISBACK(Periodic), TIMEID_PERIODIC);

which is called every 20 ms, to ensure that the windows show the same content of a file (when saved, when not saved and typing, the second split does not get refreshed). so the model here is the saved file on disk. and it clearly does not offer notification mechanism. thus, external means trigger a refresh. nice and simple. though for realtime action-triggered MVC, this approach is not quite usable. but here again, MVC is too unspecific/komplex to find a one-size-fits-all solution.

Subject: Re: MVC example
Posted by mirek on Wed, 10 Aug 2011 08:09:17 GMT
View Forum Message <> Reply to Message

Well, with some distance and more knowledge, I would say that in fact, if you look hard, you often find equivalents of all those popural terms.

E.g. think ORM. As fancy as the term sounds, in U++, it is represented by .sch file and corresponding S_table structures.

Typical dialog MVC with DB: .lay file represents view, .sch file and SqlCtrls represent model, dialog class is controller.

etc, etc.. But of course, in the same time, you can as easily operate outside those boundaries...

Mirek
