Elegant way to prevent circular events in MVC?

Asked 16 years ago Modified 15 years, 10 months ago Viewed 2k times

7

The question, in brief:

In MVC, how do you distinguish between a checkbox click (or a selectbox or listbox change) from a human meaning "Controller, modify the model", and a checkbox click (or a selectbox or listbox change) from the Controller meaning "I'm updating the view because the model has changed"?

The example:

I have a JS app (all one big HTML+JS page; there's a server behind it, and AJAX going on, but it's not important to the example) which has the notion of "Vertices" connected by "Edges". The UI lets you add and remove Vertices on a map, and enable or disable Edges between pairs of Vertices.

There are two ways to disable an Edge from Vertex A to Vertex B:

1. click on the Edge to make the "Edge Details" window provide you with a "Disable This Edge" button; or

2. click on Vertex A (or B) to make the "Vertex Details" window provide you with a checklist of nearby Vertices, from which you can uncheck Vertex B (or A).

Here's how this works under the hood in MVC (*but see the end of this post for an update, where I correct problems in my understanding*):

- Model: a list of Vertex objects and a list of Edge objects.
- View: a GMaps UI, with markers and polylines, plus checkboxes and buttons and "Vertex Details" and "Edge Details" DIVs.
- Controller:
    - JS functions that update the model when events on the checkboxes and buttons fire; and
    - JS functions that update the view when events on the models fire.

Here's the specific inelegance:

1. The user has the Vertex Details Window focused on Vertex A, and the Edge Details Window focused on the Edge from Vertex A to Vertex B.
2. The user clicks "Disable This Edge" in the Edge Details window.
3. Controller function 1 gets the click event, and calls disable() on the Edge model object.

4. The Edge model object fires the "I just got disabled" event.
5. Controller function 2 receives the "I just got disabled" event, and
    1. redraws the Edge Details Window to say "I'm disabled!" and
    2. **unchecks Vertex B in the Vertex Details Window.**
        1. **Crap!** This fires Controller function 1 again, which was listening for UI events that mean an edge was disabled!

So there's an unnecessary re-update of the Model, and re-update of the View. In a more complex view with events that fire events that fire events, this can make for dozens of extraneous updates!

Update: a great answer.

I misunderstood MVC a bit. I don't have just one View, as I described above: I have several Views into several Models. In particular, I have a checkbox-list View of Edges to a particular Node, and a separate, "detailed window-style" View of an Edge.

Furthermore, I shouldn't have one controller function updating all views when the Model changes: each View should modify *itself* when the Model changes.

So if each View registers for "state updated" events on the Model, and each View updates itself upon receipt of those events, then the answer to my circular events question is simply this:

The checkbox-list View will disable checkbox events for the moment that it is updating the checkboxes after a Model state change.

Now if a user disables an Edge via the Edge Detail window, the Controller updates the Edge Model, the checkbox-list View receives notification of the update, and the checkbox-list View is smart enough to silence checkbox events while changing the status of the appropriate checkbox.

This is much more palatable than my original solution, where one Controller updates ALL Views -- and thus has to know which views need special care and feeding to avoid loops. Instead, only the single View with troublesome UI elements has to deal with the problem.

Thanks to those who answered my question!

javascript **model-view-controller** **design-patterns** **event-handling**

edited Dec 18, 2008 at 16:34

asked Dec 17, 2008 at 3:58

Michael Gundlach
109k 11 38 41

2 Answers

Sorted by: Highest score (default)

3

Just to recap the MVC model. Views should generally update themselves. Here's how it works: a controller changes the state of the model, the model sends updates to its views, the views pull in new state from the model and update themselves. While controllers and views are generally bundled (i.e. drilling down on data in a graphic representation) they should never interact directly, only through the model. This in general of course.

So the JS functions that update your views are not actually controllers, which is an important distinction. They should be considered part of your view. This might not be helpful to the problem at hand but I thought it merited pointing out.

You can also not delete your model, I assume you mean you're deleting someting *from* your model, since no views or controllers can actually exist (or be in a functional state) if they're not backed by a model.

Not being a JS code jockey and not having used gmaps I don't really see where the problem is. Does changing the state of a checkbox(checked property) fire the onClick() event? It really shouldn't IMHO but perhaps they implemented it that way, otherwise you could just attach

your controller to the onClick() and add some logic to the checkbox (or, this being JS, in a function somewhere) to change the checkbox state. If that's not possible, option 1 and 2 are definitely your best bet.

addition: user interacting with a view

So what happens when a user wants to interact with a view? Frequently a widget will include both a view and the controller. A checkbox has a view (you can see if it's checked or not) and also a controller (you can click it). When you click the checkbox, in principle the following should happen:

- checkbox controller receives the event
- checkbox controller changes the state for the value this checkbox represents in the model
- model updates listeners (including the checkbox)
- checkbox updates its look to reflect that that value has changed

The first step, how the controller receives the event is somewhat language dependent, but in OOP languages it's likely a listener object attached to user interface events on this particular widget which either is the controller or notifies the controller of the user interaction.

edited Dec 23, 2008 at 10:56

answered Dec 18, 2008 at 9:25

wds

32.2k ● 11 ● 62 ● 84

spot on. i'll edit my question to reflect your answer. you're correct that i can't delete my model - i was confused.
– Michael Gundlach Dec 18, 2008 at 15:25

Would you mind editing your answer to include the user interacting with the view, and how the controller is notified? My problem partially lay in the fact that the same control in the view was used for both user input and feedback to the user. – Michael Gundlach Dec 18, 2008 at 15:26

Added some text, not sure if it's what you were looking for.
– wds Dec 23, 2008 at 10:56

0

This is a tough one. If I understand correctly, the problem results because you've exposed a click handler on your model, and the model's click event is caught by the controller. The controller updates the view, which in turn toggles the same event.

From my point of view, I would consider it inappropriate for the Controller to attach itself to the Edge's Click event, because it exposes too much detail about how the Edge is implemented and used. Controller does not care about how the Edge is used or any other implementation details.

In fact, canonical MVC style doesn't require the Controller to hook onto *any* Model events at all, generally because

the Model's state is not mutated by the View or any other Controllers. Its not necessary for the Model to notify the Controller that it's been changed.

To fix your problem, you should define View's interface to have a single method, such as ToggleEdge:

```
public interface GraphView
{
    event Action ToggleEdge;
}
```

Its tempting to want to create two methods, EdgeClicked and CheckboxClicked, but insisting on two independent methods like that violates the principle of encapsulation. It exposes too many implementation details to your Controller or anyone else who wants to hook onto those events. Remember, the Controller only cares that the View's state changed, it doesn't care *how* it changed.

When you implement the View interface onto your user interface, you should take care to ensure that the ToggleEdge event is invoked from *one* location. You can do this by hooking onto the Edge.Clicked event in your View and using it to toggle your checkbox; this makes your checkbox responsible for raising the Toggle vent up to the controller:

```
public class UI : UserControl, GraphView
{
    public event Action ToggleEdge;
```

```
    void OnToggleEdge(Edge edge)
    {
        if (ToggleEdge != null)
            ToggleEdge(edge);
    }

    protected void Edge_Clicked(object sender, EventAr
    {
        CheckBox chkbox = FindCheckBoxThatCorrespondsT
        chkbox.Checked = !chkbox.Checked;
    }

    protected void chkEdge_CheckChanged(object sender,
    {
        Edge edge = FindEdgeThatCorrespondsToCheckbox(
        OnToggleEdge(edge);
    }
}
```

You can make an argument that the View knows too much about its implementation now: its aware that edges and checkboxes are fundamentally connected. Maybe this is another hack, but it can probably be dismissed as "UI logic" need to keep the View's display syncronized.

Share Improve this answer

Follow

answered Dec 17, 2008 at 18:53

Juliet

81.4k ● 46 ● 199 ● 229

Princess, thanks for your response. In case it wasn't clear, my app is in JS; I've edited my question to make that more obvious. If I understand you correctly, the View is responsible for updating *itself* upon state change. Isn't the Controller supposed to update views when state changes?

– Michael Gundlach Dec 17, 2008 at 20:24

For example, your solution wouldn't work if there were three separate Views on the model. When one changed, it shouldn't be responsible for updating the others; the Controller should notice the change, then update the other views. And when it does, it accidentally fires their change events again. – Michael Gundlach Dec 17, 2008 at 20:25