The selection channel technique

Bobby Woolf

In "Making MVC more reusable,"[1] I talked about how VisualWorks has improved upon classic model-view controller (MVC). That article discussed new objects like value subviews and value models as well as the way "model" has been split into application model and domain model. Its third figure shows the objects involved when the model is split into application and domain models. When I drew it, I threw in another object that is often involved, the one that selects which domain model the application model will use. I labeled this object "SelectionChannel." In this article, I'll explain what a selection channel is.

Selection channel is an extension of what ParcPlace is calling the "Slam Dunk" architecture.[2] Slam Dunk uses subject channels to greatly simplify the code needed to connect an application model to its domain model. The idea is that once you set up an application model this way, you can insert (i.e., slam dunk) a domain model in there and everything in the application model and its window will update automatically. Selection channel takes this idea one step further to specify where the various domain models are coming from and who is inserting them into the application model.

A subject channel establishes a single point of access to a subject.

To understand selection channel, let's first take a look at where graphical user interfaces (GUIs) come from. The easiest way to produce a GUI window in VisualWorks is to use the Painter to produce a view with a bunch of widgets on it. Most of the widgets are value subviews, each displaying some value it contains: an Input Field shows some text, a Check Box shows whether a setting is on or off, etc. Where do these values come from?

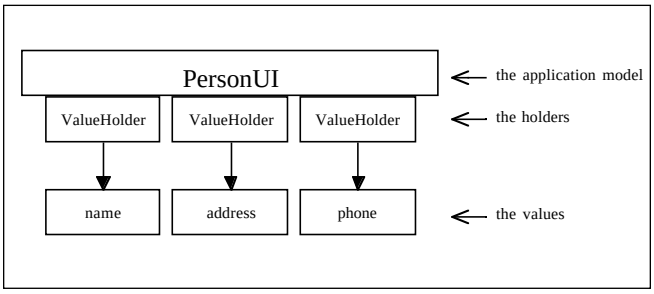

Figure 1. An application model with no domain model.

ADAPTING A DOMAIN MODEL

VisualWorks uses a revision of the MVC paradigm that breaks the model into two parts: the domain model and the application model. The *domain model* contains the information to be displayed, whereas the *application model* organizes the information so that it can be displayed. In the MVC, the window is the view and the application model is its model. This macro MVC system contains a number of MVC micro systems, one for each value subview, where the subview is the view and its model is a value model. A *value model* (an instance of ValueModel) is a simplified model that has exactly one aspect. Thus the value model contains a single value, which in this context is the information a single widget displays.

A window generated by the Painter uses an application model but, by default, does not use a domain model. The Painter generates an application model (a subclass of ApplicationModel) that is able to open the window when told to do so. Since the window contains widgets that require value models, the application model contains those value models. By default, the application model initializes those value models to be holders (instances of ValueHolder) on default values like empty string, false, zero, and nil (Fig. 1). All this default initialization is enough to make the view work: the window will open, the widgets display their values, and the widgets can be used to change their values. This, however, is not very useful functionality.

For an application model to really be useful, it should organize information that is stored in a separate object—a domain model. The question then is how the application model should get the information from the domain model. Often the first solution people discover is for the application model to copy the values out of the domain model and into the application model's value holders. This works for displaying the domain model, but if the user edits the values in the window, the process of putting the values back into the domain model is difficult and redundant. It would be easier to associate each value model with its corresponding value in the domain model.

The type of value model that knows what domain value

it contains is an adaptor value model. An *adaptor value model* (an instance of PluggableAdaptor, AspectAdaptor, etc.) extracts its value out of an object that contains the value. The adaptor calls the value's container its *subject* (accessible via ProtocolAdaptor>>subject). Each adaptor knows how to pull its value out of its subject to show the value to the user. If the user edits the value, the adaptor knows how to put the new value back into its subject. An application model adapts its domain model by using adaptors. The application model initializes its value models as adaptors instead of holders, where each adaptor's subject is the application model's domain model. This way the application model maps each widget in the window to its corresponding value in the domain model (Fig. 2).

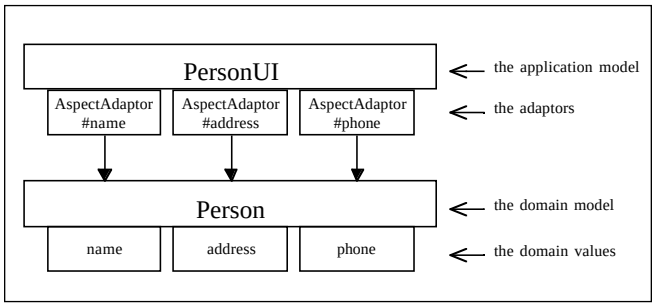

Figure 2. An application model adapting a domain model using adaptors.

SUBJECT CHANNEL

When an application model is using several adaptors to adapt its domain model, all the adaptors must have the same subject. This way all the widgets in the window are showing different parts of the same domain model. This presents some difficulty when setting up the adaptors on a domain model: the code must ensure that all the adaptors are connected to the same domain model. Furthermore, to change to a different domain model, all the adaptors must be disconnected from the old model and connected to the new one. This leads to verbose, repetitive code that is prone to flaws.

The code for connecting and disconnecting a set of adaptors to a single subject can be virtually eliminated using a subject channel. A *subject channel* (accessible via ProtocolAdaptor>>subjectChannel) establishes a single point of access to a subject. The subject channel is itself a value model, typically a holder, whose value will be used as a subject. When multiple adaptors share the same subject channel, they are all guaranteed to share the subject. Even if the subject is replaced with a new one, the subject channel will automatically cause all the adaptors to disconnect from their old subject and connect to their new one. This way the application model does not need any code for connecting or disconnecting its adaptors to their subject; the subject channel feature does this automatically (Fig. 3).

Note that in VisualWorks 2.0, PluggableAdaptor does not support subject channel because it is not a subclass of ProtocolAdaptor.

The application model, instead of connecting its adaptors to their subject, has to connect them to their subject channel. First it must set up its domain model for use as a subject channel. Then it must connect the adaptors to their subject channel.

There are two ways an application model can set up the subject channel. One alternative is for the application model to obtain a value model that contains the domain model. The other alternative is for the application model to obtain the domain model and wrap a value model around it. Either way, once the application model has its domain model in a value model, it stores the value model. This value model becomes the application model's *domain model channel*, which means that the value model is the application model's single point of access to its domain model. The application model will go through its domain model channel to get its domain model. Once the domain model channel is established, the channel object (the value model) should never be replaced with another.

The application model uses its domain model channel to connect all its adaptors to the same domain model. As the code in the application model creates its adaptors, rather than setting each adaptor's subject to be the domain model, it sets each one's subject channel to be the domain model channel. This way the subject channels and the domain model channel are all the same object. Each adaptor sees this domain model channel as its subject channel and will connect itself to its subject via the subject channel. This way, if the domain model is replaced with a new one, all the adaptors will attach themselves to the new one.

SELECTION CHANNEL

Subject channel does not specify where the subject comes from, just that it will become available by being inserted into the subject channel. However, somebody must be setting the subject channel to contain a new subject. *Selection channel* considers not only that the subject is being replaced with a new one, but what object is making the replacement and where it's getting the new subject from. Whereas a subject channel just specifies how the subject can be accessed, selection channel also specifies where new subjects come from (Fig. 4).

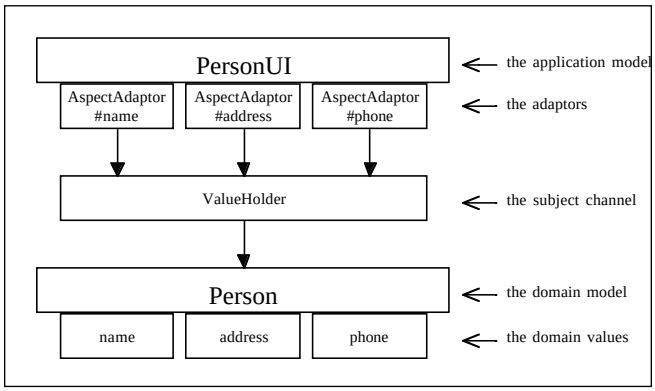

Figure 3. An application model adapting a domain model via a subject channel.

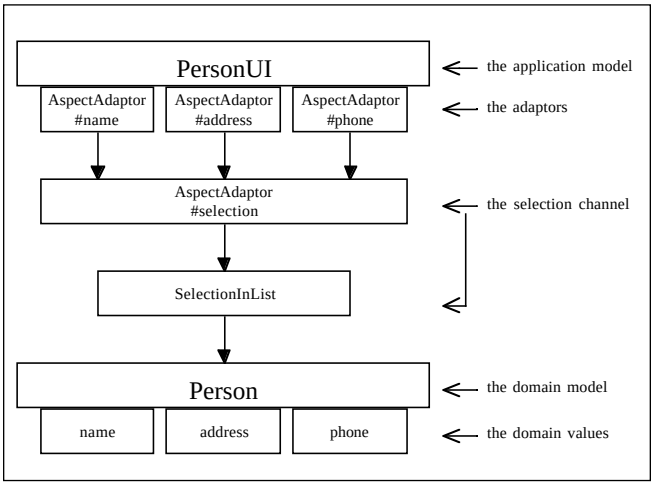

Figure 4. An application model adapting a domain model via a selection channel.

Considering how new subjects are obtained is important for setting up the subject channel correctly. For two objects to share the same value using a value model, they must share the same value model.[3] Thus the object obtaining the new selection and the objects using the selection must share the same value model. The selecting object, the one choosing a new subject from a list of possibilities, must store its selection in a value model. This selection value model will be shared by the other objects using the selection.

VisualWorks already includes an object that is perfectly suited for use as a selection channel, the selection in list (an instance of SelectionInList). A selection in list itself is not a selection channel, but it contains one. In VisualWorks 1.0 you had to create your own selection channel on a selection in list using a pluggable adaptor. VisualWorks 2.0 has an enhanced aspect adaptor and a new method, SelectionInList>>selectionHolder, which returns an adaptor that is perfect for use as a selection channel. Selection channel does not require a selection in list, but that is the object most commonly used for this task.

An application model that will use a selection channel must obtain it from the selection object so that they will be sharing the same one. When an application model is establishing its domain model channel, it can either obtain that channel or obtain the domain model and create the channel itself. However, if the domain model channel is a selection channel, they must be the same object, so the application model must not create a new channel. If it does, the application model will not be notified when the selecting object changes the selection.

CONCLUSION

In a nutshell, here's what the article said:

- An application model organizes a group of values so that they can be displayed by the widgets in a window.
- The group of values should be stored in a domain model.
- An application model adapts its domain model using adaptor value models.
- An application model should use a domain model channel to set its adaptors' subject channels; this is a simple way to force them all to adapt the same subject/domain model.
- This domain model channel should be a selection channel from a selecting object such as a selection in list; this way the window will automatically display the item selected in the list.

Hopefully this explains what that object labeled SubjectChannel is all about.

References

1. Woolf, B. Making MVC more reusable, THE SMALLTALK REPORT 4(4):15–18, 1995.
2. Robicheaux, M. Visual Slam Dunk tutorial, THE PARCPLACE INTERNATIONAL USERS CONFERENCE, 1994.
3. Woolf, B. Understanding and using the ValueModel framework in VisualWorks Smalltalk, PATTERN LANGUAGES OF PROGRAM DESIGN, Coplien, J.O. and D.C. Schmidt, Eds., Addison-Wesley, Reading, MA, 1995.

Bobby Woolf is a Member of Technical Staff at Knowledge Systems Corp. in Cary, North Carolina. Comments and questions are welcome at woolf@acm.org.