Pictures from the Object Explorer for VisualWorks™

Kent Beck, First Class Software, Inc.

Copyright © 1994, First Class Software, Inc. All rights reserved.
Permission is hereby granted for unlimited non-commercial duplication

Introduction

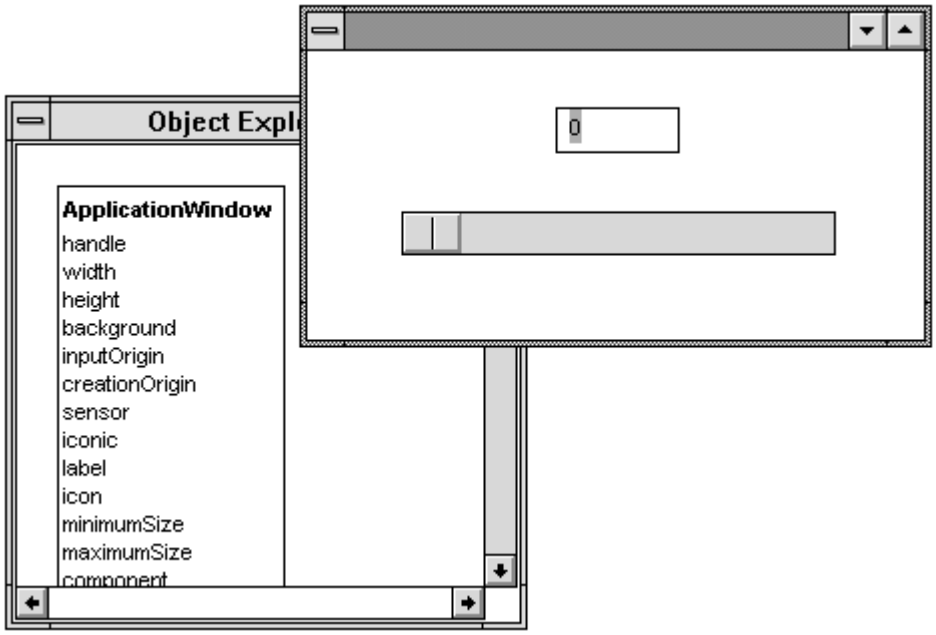
This is a short paper that graphically demonstrates the power of the Object Explorer to explain complex objects. The example that follows shows the Object Explorer quickly presenting how complex communities of objects work together. The Object Explorer can do this for your objects, too: give you a better understanding of how they work together and communicate that understanding to others. The example is also interesting because it describes objects that are presented sketchily, if at all, in the ParcPlace documentation.

The example shows how VisualWorks objects work together to present and manage information on the screen. The GUI builder in VisualWorks is supposed to protect you from ever having to see these objects, but all the customers I talk to have had the frustrating experience of trying to understand them with just the standard Smalltalk tools.

The two families of objects you need to understand if you want to understand VisualWorks are VisualComponents, which display, aggregate, and modify visual information, and ValueModels, which simplify the protocol between views and models. This paper describes both families using many of the features of the Object Explorer.

Example

I'll use the SimpleSlider for my sample objects. Here is what it looks like on the screen and with the beginning of an Object Explorer (I got the Object Explorer by interrupting the slider window and "exploring" from the resulting debugger):

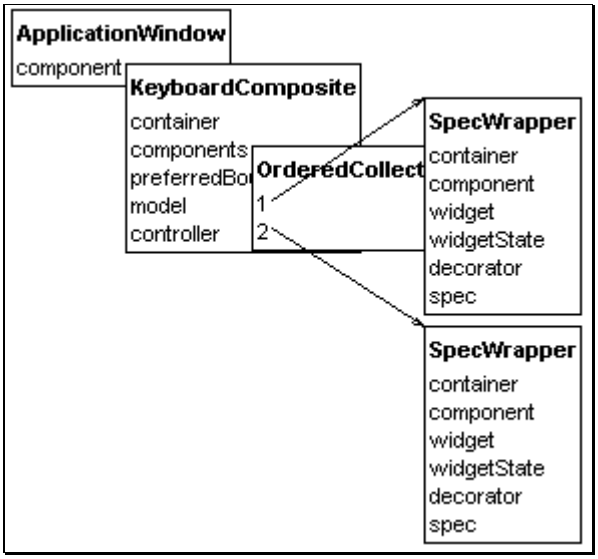


VisualComponent

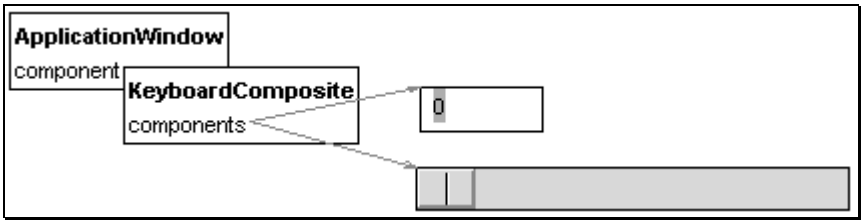
Now we can begin to look at the objects behind the slider. The visual objects in VisualWorks divide into three categories:

- objects that display things (Views if they have to offer interaction, pure VisualComponents otherwise)
- objects that lay out VisualComponents (Composites)
- objects that modify the appearance or behavior of VisualComponents (Wrappers).

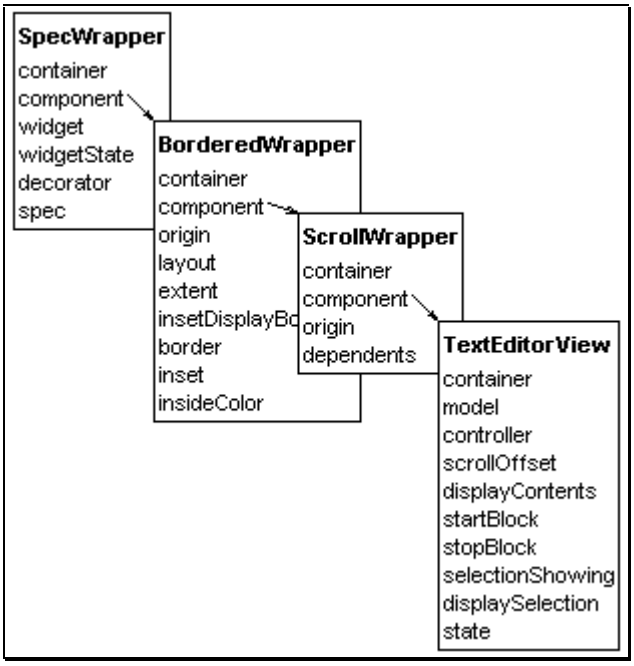
The following picture shows the top level window (ApplicationWindow, which acts as a wrapper) containing a composite that manages where typed input goes and defines tab ordering (the KeyboardComposite), and two wrappers on VisualWorks-constructed components (the SpecWrappers). At this point the utility of the standard Smalltalk inspectors would be exhausted. If you were using the inspector, you'd have five windows on the screen, and you wouldn't know which SpecWrapper was which, or even if they were different objects.



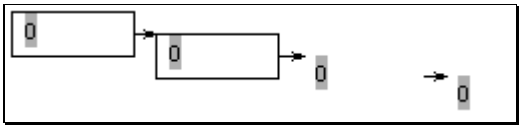
One of the benefits of creating a visual environment for showing instances is that we can use visuals to enhance the presentation of information. First, notice that the two SpecWrappers are different objects. If the OrderedCollection referred to the same object twice, we'd have two pointers to the same SpecWrapper. This feature will be indispensable when we start looking at how the two views in the slider stay synchronized. Second, because we have a visual environment, and because Smalltalk is so dynamic, we can have the objects describe themselves visually. By changing the SpecWrappers to immediate format, we can instantly tell which is which.



One of the most powerful, but most elusive parts of the design of VisualWorks is how the wrappers work together with the leaf components (Views, in this example) to change their appearance and behavior. Here is a closer look at the text editor:



By changing the objects in this picture to immediate format, you can see the effect of the BorderedWrapper:

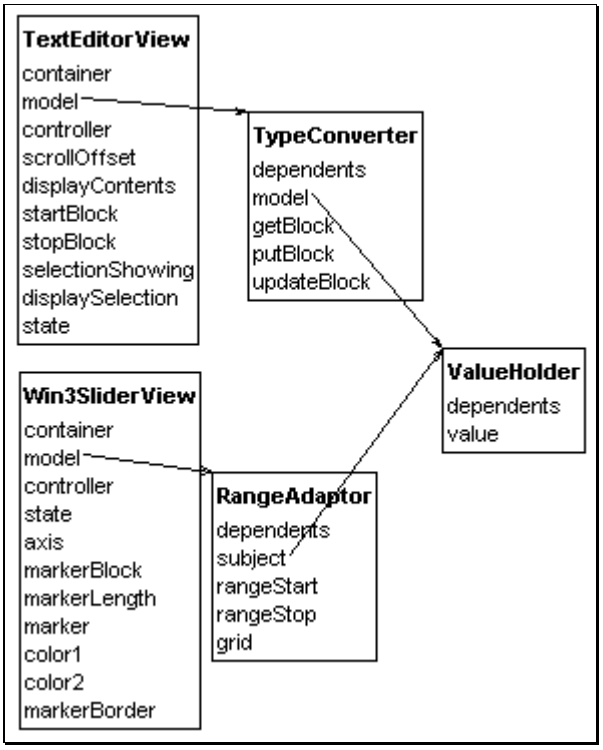


The effect of the other two wrappers is not visible. The SpecWrapper doesn't have a visible effect because its responsibility is not to change the appearance of its component, but to reconstruct its from the specification created in VisualWorks. After this reconstruction happens, it passively passes on display messages to its component (until a new reconstruction is requested, for example by requesting a different look and feel from preferences).

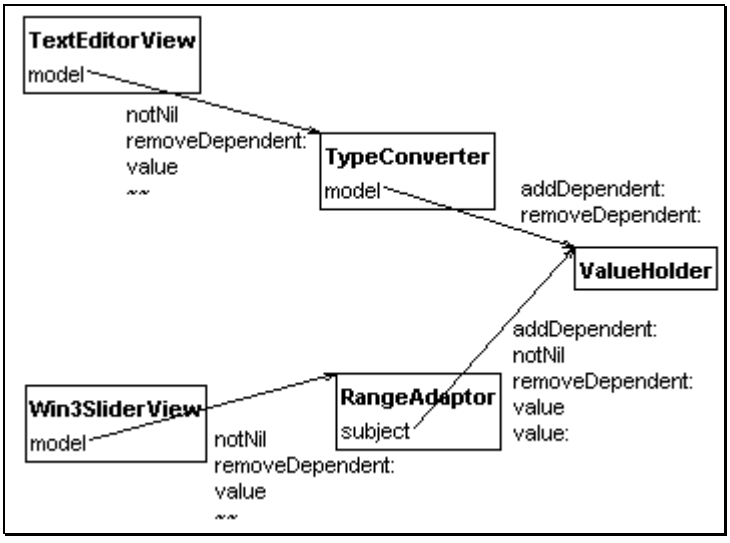
The ScrollWrapper won't ever have any effect in this example, because there is no scroll bar to change its origin.

ValueModels

Now I want to show how the two views stay synchronized. First I'll show how they both display the same data. Back in olden times, the model of the view was the information the view displayed. The conversation between a view and its model was fairly rich. Now, filters are put between the view and the model to simplify the protocol. Here, the TextEditorView and the Win3SliderView both have the same ValueHolder as a model, but each relies on another object (generically a ValueModel) to convert the raw information to something digestible to the view.



We can use another feature of the Object Explorer, contracts, to show just how simple the protocol between view and model is. A contract is the set of messages one object sends to another. The Object Explorer can show the contract an object has with an instance variable, like this:

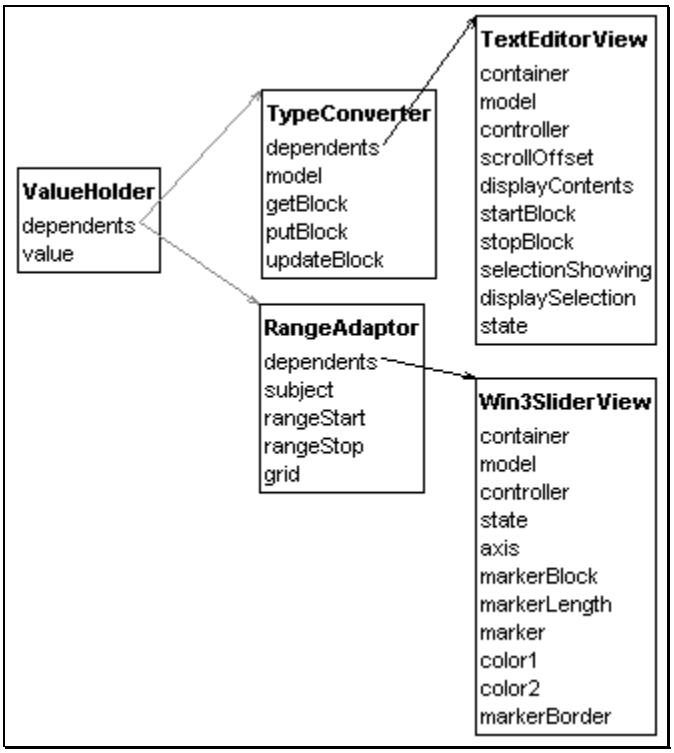


In this picture, the references from object to object have been annotated with the messages that flow along them. As you can see, the only message of substance which the views require of their model is "value". Similarly, the RangeAdaptor only relies on value and value: of its subject. The TypeConverter doesn't show the message value because it is parameterized with a block which is passed the model. The model is sent "value" inside the getBlock, which is outside the scope of the code I wrote to reverse engineer contracts.

Notice in this picture how the irrelevant variables have been eliminated. The Object Explorer has many such features to help you simplify your presentation of objects. This makes it easy to break one explanation up into a series of diagrams, rather than insist your reader try to understand hundreds of objects and references at once.

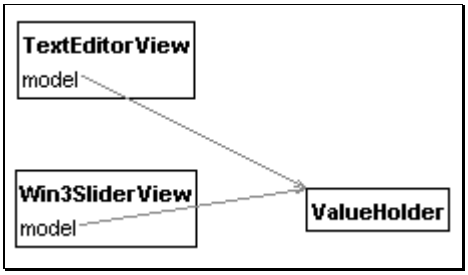
Now we've seen how the views get the same value. Each requires the value in a different form. The TextEditorView expects a String, which the TypeConverter creates from a Number. The Win3SliderView expects a Float between 0 and 1, created by the RangeAdaptor from its rangeStart and rangeStop.

How do the two views stay in sync? When one of them changes the ValueHolder, how does the other one update? The following picture shows how the updates are propagated. Notice that the references from the ValueHolder's dependents to the TypeConverter and RangeAdaptor are gray, showing that there is an intervening object (in this case a DependentCollection) that has been hidden.

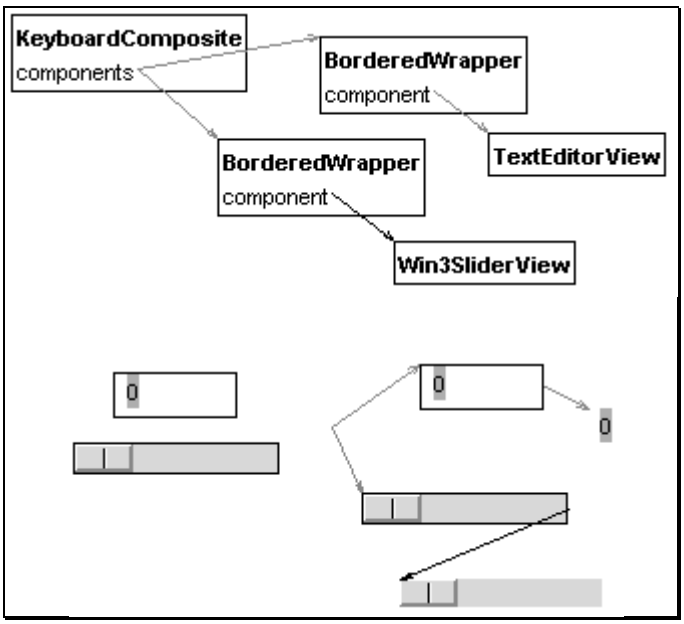


Conclusion

This concludes our quick tour of the mechanisms behind VisualWorks. The two key objects you need to understand if you want to know how VisualWorks displays information are ValueModels and VisualComponents. The Object Explorer is indispensable at showing both. Resolving multiple pointers to a single object visually, eliding intervening objects, and contracts make the presentation of models based on ValueModels simple. Here is a simple diagram that I call "Multiple Views Indirectly Share a Model":



The VisualComponent hierarchy, and the roles of components, wrappers, and composites, are directly shown by the Object Explorer's immediate object format. Here is a simple diagram I call "Component, Wrapper, Composite":



The Object Explorer for VisualWorks is available for $499 per user. It comes in vanilla VisualWorks and ENVY library formats. Volume and source code licenses are also available. If you have comments about this paper, or would like more information about the Object Explorer for VisualWorks, please contact me at:

Kent Beck
First Class Software, Inc.
P.O. Box 226
Boulder Creek, CA 95006

v: 408.338.4649
f: 408.338.1115
e: 70761.1216@compuserve.com

Object Explorer and Object Explorer for VisualWorks are trademarks of First Class Software, Inc.
VisualWorks is a trademark of ParcPlace Systems, Inc.