Dwight Deugo Wayne Beaton

# Reusable components

USING A GRAPHICAL user interface builder like VisualWorks' Canvas tool and ObjectShare's Window Builder Pro, a developer avoids the tedious task of hand coding an application window's layout by using a tool that generates the code after the window is constructed visually. Using such GUI builders, developers view a window's appearance when opened and how it appears when resized before ever executing a line of the application window's "real" code. These tools permit a developer to identify what application methods should execute when specific events, such as the clicking of a button, occur in the application window. Good ones will write method stubs that developers subsequently complete to perform the required reaction to given events. Reaction methods often send messages to business objects to set or retrieve information about them and may force the window to update its views. Although these tools no longer require that developers hand code an application window's layout, they still require them to implement the interaction between the views and the underlying business domain.

The latest generation of GUI builders include tools such as VisualSmalltalk's PARTS Workbench and IBM's VisualAge Composition Editor. Both GUI builders are examples of the new Construction from Parts technology called *Visual Programming*. They enable developers to create windows and other components by assembling and connecting reusable components, also known as *parts*. Rather than hand-coding the interactions between the parts, developers make visual connections between a source part's events and another part's actions. The GUI builders still write Smalltalk code to construct the window for the developer, but, in addition to layout code, they also write code connecting events on parts to the execution of methods on others. Provided that there are good parts, developers do not need to write any Smalltalk code to develop their application windows. They just assemble them from existing parts and say "go". However, without guidelines, it is now possible to paint spaghetti instead of just writing it!

Dwight Deugo and Wayne Beaton are senior members of the development educational staff at The Object People, in Ottawa, ON, Canada. Dwight (dwight@objectpeople.on.ca) has immersed himself in objects for more than 10 years and has helped clients with their object immersions as a project mentor and as a course instructor. Wayne (wayne@objectpeople.on.ca) is the coordinator of course construction and a software developer.

The success of visual programming depends on how organizations use it and on the availability of a rich library of reusable generic and domain-specific parts. This column will focus on visual programming tips and techniques to help you become a more effective visual programmer. Future columns will cover how to manage the number of connections in your window and describe visual debugging techniques. We will also provide many examples of reusable components developed using visual programming parts and techniques, such as an advanced factory part, a broker, a marquee, and web parts. Initially, we will use examples derived from IBM's VisualAge for Smalltalk environment, but we will include examples from ParcPlace-Digitalk's next product release.

This column describes the building blocks for constructing any application window: parts and connections. As an example, we build an Action List Window with IBM's VisualAge using only those building blocks—no Smalltalk code. The window's requirements are to let a user enter any number of actions into a To-Do list and then move them to a Completed list. Our goal is to demonstrate that, when given the building blocks and good reusable components, you can do a substantial amount "programming" without writing a single line of code. Be warned that visual programming rarely, if ever, provides a complete solution.

## REUSABLE COMPONENTS (PARTS)

Before one can do any visual programming, one must have access to, or must create, a number of reusable components (parts). There are two different types of parts: *Visual* and *Nonvisual*. Visual parts have visual representations and appear in a runtime application window, for example, buttons, lists, input fields, and labels. Nonvisual parts have no visual representation, such as a Printer, CD player, Ordered Collection, Variable, and any domain-specific business parts. Nonvisual parts implement objects that provide logic, storage, and resource access for your application windows. Visual and Nonvisual parts are simply assemblies of visual and nonvisual parts.

In VisualSmalltalk's PARTS, all parts in the Workbench are instances of Smalltalk classes. The part's default interface includes all the messages the Smalltalk object understands and all of the events it can trigger. In IBM's VisualAge, all parts in the Composition Editor are Smalltalk classes. The part's default interface is empty until the

developer decides what portion of the part's Smalltalk class' interface to make public.

A part's interface includes attributes, events, and actions. Attributes represent properties of a part, such as the name of an employee, that other parts access. An attribute can be any Smalltalk object, including other visual and nonvisual parts. One can initialize a part's attributes using a GUI builder's property or settings tool at development time or can access them dynamically at runtime. Actions are an operation that a part executes when events on other parts trigger them. For example, a button click event (generated when the user clicks on the button in the application window) could trigger a window's close action. Actions correspond to Smalltalk methods or code fragments. Events are signals that one part can send to another to notify it that something has occurred.

## CONNECTIONS

A developer specifies relationships between parts by making connections between them. The first type of connection is an event-to-action connection. This link connects an event of one part with the execution of another part's action. When the event triggers, the action executes. The second type of connection is an attribute-to-attribute connection, which can be viewed as a two-way event-to-action connection. The change of one part's attribute (the event) triggers the setting of the second part's attribute to the same value (the action), and vice-versa.

A link, also called connection, is a type of part. Therefore, it has attributes and events. The attributes of a connection correspond to the parameters that the action at the end of the connection requires and the action's result. If an action requires no parameters, the connection has only one attribute: a result. Since actions just execute Smalltalk code, the connection stores the result object as an attribute. Since the setting of an attribute is equivalent to an event, it is possible to make a connection between the result event and other actions. One can trigger an action on another part when a previous action finishes and returns a result.

Events may or may not generate parameter values. For example, the clicking of a button only triggers a click event. On the other hand, the selection of an item in a list generates a selection event and provides the selected object as an argument for a connection to use as one of its parameters. Of course, one can change the value by making a connection to the link.

## VISUAL PROGRAMMING EXAMPLE

Visual programming permits developers to quickly construct application windows provided the appropriate parts are available. Using VisualAge for Smalltalk version 3.0, we quickly constructed the "ActionListWindow" shown in Figure 1. This window allows the user to construct a list of actions to do for the current day. From that list, completed actions can be moved to a completed action list. At the end of the day, the user should have all of his or her actions in the completed actions list (ha ha)!

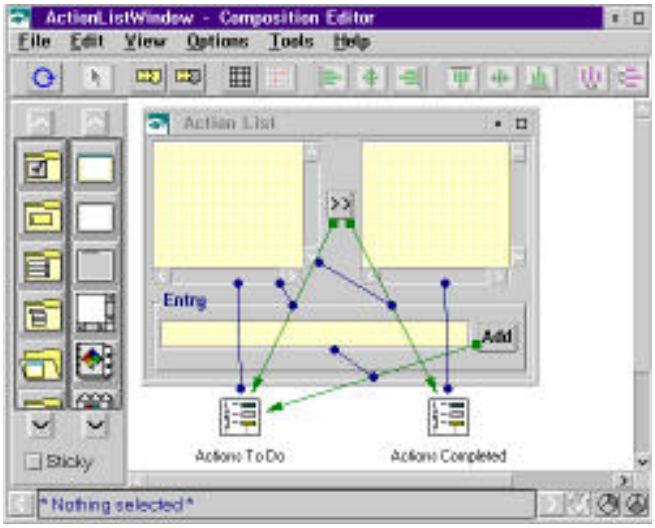


Figure 1. ActionListWindow in the Composition Editor.

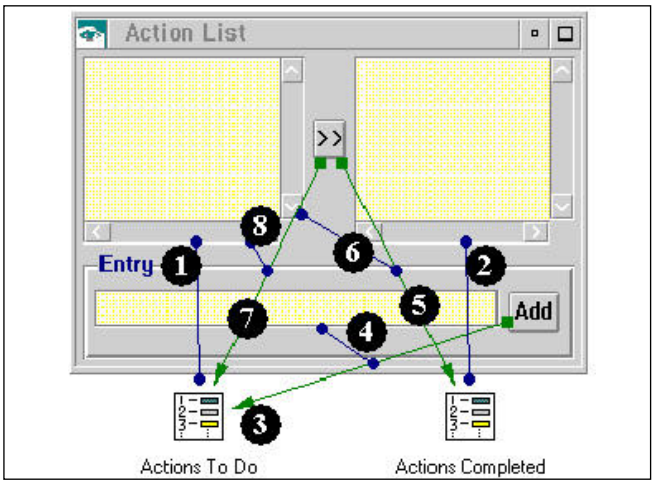


Figure 2. ActionListWindow Connections.

This first-pass of the ActionListWindow contains several visual parts, two nonvisual parts, and a few connections. The ordered collection* part, "Actions To Do," is connected to the left-most list with an attribute-to-attribute connection. The collection's "self" attribute† is connected to the list's "items" attribute. This connection specifies that the ordered collection stores the items to display–if the ordered collection changes in any way, the change is automatically reflected in the list box. A similar connection links the "self" attribute of the ordered collection titled "Actions Completed" to the "items" attribute of the right-most list.

The "clicked" event of the push button labeled "Add" is connected to the "add:" action of the "Actions To Do" ordered collection through an event-to-action connection. The "add:" action requires a parameter. We specify the connection parameter with an attribute-to-attribute connection from the "anObject" *continued on page 28*

* An ordered collection holds any number of Smalltalk objects in the order in which they are added.

† In VisualAge, all parts have a "self" attribute. This attribute represents the whole part.

Link # PartName.attribute/event/action → PartName.attribute/event/action

1. ActionList.items → 'Actions To Do'.self
2. ActionsCompletedList.items → 'Actions Completed'.self
3. AddButton.clicked → 'Actions To Do'.add:
4. inputField.object → connection3.anObject
5. >>Button.clicked → 'Actions Completed'.add:
6. ActionsCompletedList.selectedItem → connection5.anObject
7. button.clicked → 'Actions To Do'.remove:
8. ActionsCompletedList.selectedItem → connection7.anObject

Figure 3. ActionListWindow Legend.

*continued from page 25* attribute of the original connection to the "object" attribute of the entry field. These two connections provide the ability to add objects to the ordered collection. Any objects added are automatically displayed by the connected list.

Clicking the "move" button, labeled ">>," moves the selected item from the left-most list to the right-most one. Objects removed from the "Actions To Do" ordered collection are added to the "Actions Completed" ordered collection. The order of the following connections is important.‡ The "clicked" event of the "move" button is connected to the "remove:" action of the "Actions Completed" ordered collection. This event-to-action connection requires an object (the object to be removed) that is supplied by connecting the "anObject" attribute of the event-to-action connection to the "selectedItem" attribute of the left-most list box. The "move" button's "clicked" event is also connected to the "remove:" action of the "Actions To Do" ordered collection, with the "anObject" parameter supplied again by the "selectedItem" attribute of the left-most list box.

Clearly, we require a better way of describing the connections—textual descriptions are too long. A concise connection representation is both desirable and necessary. Figure 2 shows our Action List Window again, but this time we have added line labels (unfortunately VisualAge does not provide this facility for us) and Figure 3 shows the legend.

‡ Once you remove an object from an "Action To Do" ordered collection, it is no longer in the left-most list. Therefore, it can no longer be the selected item and cannot be moved to the "Actions Completed" ordered collection.

## IN THE FUTURE

To keep this example small, we have avoided certain issues. The push button is not disabled when its does not apply. The "move" button should be enabled only when there is a valid selection in the left-most list box. The "Add" button should be enabled only when the user has entered data in the entry field. Perhaps some ability to remove items from one or both lists might prove useful. Ultimately, the information needs to persist in some way. These are issues we intend to address in future columns.

## THE CODE

The code used in this column is available on the World Wide Web. Our URL is http://www.objectpeople.on.ca.