Chapter

# 27

# Extending the Application Framework

The basic VisualWorks ApplicationModel framework is an excellent way to approach building an application with user interface screens. However, as with most things, it's possible to extend the framework to make the development of user-interface oriented applications easier. In particular, Tim Howard has written extensions to the ApplicationModel framework that he has described in various articles and in a book.

The extensions consists of several new classes: *ExtendedApplicationModel*, *ExtendedSimpleDialog*, *DomainObject*, *DomainAdaptor*, and *CollectionAdaptor*. The code for all these extensions is in the Smalltalk Archives. It's worth spending some time looking into using these extensions in your application.

## Advantages of the extended framework

If you subclass your applications off *ExtendedApplicationModel* rather than ApplicationModel, you get a lot of benefits in terms of cleaner and simpler code. Let's look at a few of them here.

### *Simpler code*

ExtendedApplicationModel allows you to do everything you could do with ApplicationModel, but makes much of it a lot easier. Here are some examples written first with ApplicationModel functionality, and second with ExtendedApplicationModel functionality.

**ApplicationModel**

```
(self builder componentAt: #saveAB) enable.
(self builder componentAt: #editAB) enable.
(self builder componentAt: #cancelAB) disable.
```

**ExtendedApplicationModel**

```
self enable: #(saveAB editAB).
self enable: #(saveAB editAB) disable: #cancelAB
```

**ApplicationModel**

```
component := self builder componentAt: #employeeNameIF.
widget := (self builder componentAt: #employeeNameIF) widget.
```

```
controller := (self builder componentAt: #employeeNameIF) widget controller.
```

**ExtendedApplicationModel**

```
component := self component: #employeeNameIF.
widget := self widget: #employeeNameIF.
controller := self controllerFor: #employeeNameIF.
```

## *Eliminating instance variables*

ApplicationModel keeps an instance variable for each widget on the screen that will be handling input. Each input field, each list, each table, and so on will require at least one instance variable. However, the builder also keeps track of the data, so the instance variables are actually redundant. ExtendedApplicationModel uses the builder's variables, allowing you to write applications without all the usual instance variables. To do this, it provides new methods for creating ValueHolders and SelectionInLists. Here is an example of an access method using the new functionality.

```
colors
    ^self
        selectionInListFor: #colors
        collection: self domain colors
        selectionChange: #colorChanged
```

Notice that we are not initializing or returning an instance variable. The code is also smart enough to return the SelectionInList if it exists, and to create a new one if it doesn't exist. It also registers as a dependent of the SelectionInList, removing the necessity to set up the dependency using **onChangeSend:to:**.

## *DomainAdaptors*

Recall that in Chapter 23, Model-View-Controller, we noted that the ApplicationModel framework provides the concept of a *subject channel* which makes it possible to replace one domain object with another and have all the AspectAdaptors now refer to the new object. The *DomainAdaptor* class provides additional mechanisms for associating screens with domain objects. A DomainAdaptor is in effect an editor for a single domain object. You can open a DomainAdaptor on a domain object, or simply open a DomainAdaptor, which creates a new domain object. DomainAdaptors allow you to replace a domain object with a new domain object, and they update the data in the views when you do so.

```
employeeDA := EmployeeUI open.
employeeDA := EmployeeUI openOn: anEmployee.
employeeDA domain: anotherEmployee.
```

DomainAdaptors make it easy to create AspectAdaptors using no instance variables, and to register for change messages at the same time. For example,

```
name
    ^self aspectAdaptorFor: #name changeMessage: #nameChanged
```

They also let you edit collections that are part of the domain object. For example, if the employee has a set of skills that you want to add to or remove from as part of editing employee information, your **skills** method might look like the following.

```
skills
    ^self
        collectionAdaptorFor: #skills
        addSelector: #addSkill
        removeSelector: #removeSkill
        changeMessage: #skillSelectionChanged
```

### *Other useful mechanisms*

The ExtendedApplicationModel allows you to keep track of your parent application, which gives you the opportunity to send it messages if necessary. You can open an application as a single instance, which ensures that only one instance of the application will be created. If you try to open the application again, the single instance will be expanded or raised if it was collapsed or behind another window. There are extensions to Dialogs, and applications can be opened either as a Dialog (modal) or as a Application window (non-modal). There are many, many more features than we've discussed here; this chapter simply touches on the extensions to the ApplicationModel framework.

## How to get the extended framework

Tim Howard has written articles on his frameworks extensions in several issues of The Smalltalk Report, and his code is available in the Smalltalk archives (the file *domain.st* contains all the classes mentioned above). For information about retrieving software from the Smalltalk Archives, see Chapter 35, Public Domain Code and Information. However, I recommend that you purchase his book, *The Smalltalk Developer's Guide to VisualWorks*.

Tim Howard
The Smalltalk Developer's Guide to VisualWorks

SIGS Books, ISBN 1-884842-11-9.
Prentice Hall, ISBN 0-13-442526-X

The book covers the same material as his articles in The Smalltalk Report plus a lot more. It comes with a diskette containing all the framework extensions and a significant amount of example code. In particular, it contains an excellent tutorial that is run as a Smalltalk application, and which is invaluable in learning the VisualWorks application framework and application widgets, as well as Howard's extensions to the framework. To me, the tutorial itself is easily worth the price of the book.