Developing User Interfaces

Dan R. Olsen, Jr.

Sponsoring Editor	Michael B. Morgan
Production Manager	Yonie Overton
Production Editor	Julie Pabst
Editorial Coordinator	Marilyn Uffner Alan
Copyeditor	Jeff Van Bueren
Text Design	Side by Side Studios
Illustration	Cherie Plumlee
Composition	Nancy Logan
Cover Design	Ross Carron Design
Proofreaders	Erin Milnes, Gary Morris
Indexer	Steve Rath
Printer	Courier Corporation

Morgan Kaufmann Publishers, Inc.
Editorial and Sales Office:
340 Pine Street, Sixth Floor
San Francisco, CA 94104-3205
USA

Telephone	415/392-2665
Facsimile	415/982-2665
Email	*mkp@mkp.com*
WWW	*http://www.mkp.com*
Order toll free	800/745-7323

Library of Congress Cataloging-in-Publication Data
Olsen, Dan R., 1953–
Developing user interfaces/Dan R. Olsen, Jr.
p. cm.
ISBN 1-55860-418-9
1. User interfaces (Computer systems) 2. Computer software—Development. I. Title.
QA76.9.U83043 1998
005.4'28--dc21 97–45231

To m
and a ga

Chip

A Chip is a simple object that consists of the following:

`CenterPoint`
Center of the chip in the layout.

`Name`
Name of the chip.

In this simple application, the class Chip has no methods of its own. The entire functional behavior is captured in the Circuit class. In general, this would not be true. Circuits would consist of a variety of classes of circuit objects, each of which would have its own behavior. We will discuss more complex models in later chapters when we have more powerful geometric and architectural tools to handle them.

Wire

Wires are also quite simple and contain only their relevant data, as follows:

`Chip1`
Chip index to which the wired is connected.

`Connector1`
Connector index in Chip1 to which the wire is connected. All Chips have exactly 8 connectors.

`Chip2`
Chip index for the other end of the wire.

`Connector2`
Connector index from Chip2 for the other end of the wire.

5.2 Model-View-Controller Architecture

The Smalltalk system was developed as a language and an environment for building interactive applications.[1] As part of that development, an architecture for interactive applications was designed. This object-oriented approach was called the model-view-controller (MVC) architecture.[2] A schematic of this architecture is shown in Figure 5-2.

The *model* is the information that the application is trying to manipulate. This is the data representation of the real-world objects in which the user is interested. In our logic diagrams, the model would consist of the Circuit, Chip, and Wire classes.

The *view* implements a visual display of the model. In our application, there are two views, the circuit view and the part list view. Anytime the

Figure 5-2

model is
change tl
screen th
aged. Wh
the displ
some sys
will use
maintain
A mod
be notifi
Later, wh
be redraw
and by an
based on
is also th
The *cc*
what they
the contr
the curre
lated. Th
objects ar
for positi
pass a mo
a wire, or
needs, it
changes.
notify the
Becaus
twined ar
many arcl
Chapter

n. The
al, this
circuit
s more
ric and

llows:

Chips

it for
itec-
oach
ic of

late.
er is
cuit,

ion,
the

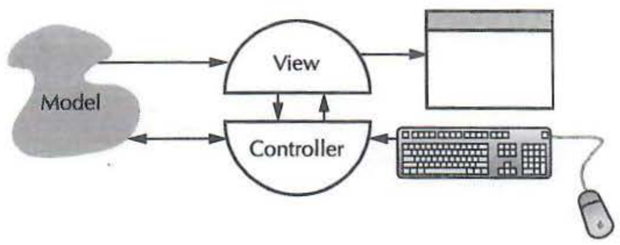

Figure 5-2 Model-view-controller

model is changed, each view of that model must be notified so that it can change the visual presentation of the model on the screen. A region of the screen that is no longer consistent with the model information is called *damaged*. When notified of a change, the view will identify the changed parts of the display and report those regions as damaged to the windowing system. In some systems, such regions are called invalid or out of date. In this text, we will use the term damaged. Reporting of damaged regions is fundamental to maintaining views on the screen.

A model, like ours, may have multiple views. In such a case, all views must be notified of the changes and the windowing system will collect them all. Later, when the main event loop looks for a new event to process, there will be redraw events waiting for any views that were affected by damage reporting and by any windowing operations. Each view must redraw the damaged areas based on information in the model. In addition to drawing the display, a view is also the location for all display geometry as will be discussed later.

The *controller* receives all of the input events from the user and decides what they mean and what should be done. In the circuit view of our example, the controller would receive a mouse-down event and must determine from the currently selected menu item whether wires or chips are to be manipulated. The controller must communicate with the view to determine what objects are being selected. For example, since the circuit view is responsible for positioning all of the chips in the window, the controller must be able to pass a mouse point to the view to determine if that mouse point is over a chip, a wire, or in empty space. Once the controller has all of the information that it needs, it will make calls on the objects in the model to make the appropriate changes. These calls by the controller on the model will cause the model to notify the views, and the displays will be updated.

Because the functionality of the controller and the view are so tightly intertwined and also because controllers and views almost always occur in pairs, many architectures combine the two functions into a single class. Recall from Chapter 4 the WinEventHandler class, which had several methods for responding to events. The Redraw method would implement the majority of the view. (The methods to handle notification from the model and object selection for the controller must be added.) The mouse and keyboard methods would implement the controller functionality. The model is implemented based on our functional design as described in Chapter 2.

5.2.1 The Problem with Multiple Parts

In simple applications, it is tempting to combine the model, view, and controller into a single class or into global variables. Such an approach will not scale up to large applications. The model classes must be separated out for two reasons. The first is that there may be multiple models that a user is working with. In our example, the user may have an old version of the circuit on the screen and may be using it as a guide to design a new version in a separate window. This scenario would require multiple models and multiple views. The implementations would be the same but different information is being manipulated in each case.

A second problem, which is frequently ignored by those building simple applications, is the fact that a model may have more than one view. In our example, the model has at least two views, the circuit view and the parts list view. Each view is very different but each must be updated when a chip is added to the circuit. There may also be multiple, similar views of the same model. Our example application does not support scrolling of the circuit view, but let us suppose that it did. Let us also suppose that the circuit was very large and the user had need to work in two separate areas of the circuit at once. An additional circuit view of the same circuit could be created at run time. Each view could be scrolled to a different part of the circuit. In such an application, there can be any number of views of the same model, depending on what the user is trying to do. Each of these views must be kept consistent with the model and the user must be able to interact with the model through the controllers of each of those views. The support for multiple views is the primary reason for the separation between the model and the view-controller.

There are also software maintenance reasons for the separation. Suppose, for example, that our users look at our first implementation and decide that it is important to have a wiring list view that shows all of the wires and that names their connections. We could implement the new view and its controller and add it to the list of views that need to be notified whenever the model changes. The existing views would not need to be changed and the model would be unaffected. With the addition of a new view, new model information may be needed; however, the old views would still respond in the same way.

Suppose that our graphics designers and marketing people decide that chips should be drawn with a 3D look rather than a flat schematic look. Only the

Figure 5

view w
would
way, b
That i
troller.
The pa
pender

5.2.2

In mos
model
to upd
betwe
relatio
to the
Let
sists of
geome
shapes
display
vertica
One
moved
will er
positio
shown
It w
results

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | sales@docketalarm.com | 1-866-77-FASTCASE

fastcase®
Smarter legal research.