Advanced Object Oriented Programming Mandatory Assignment 1

Magnus Erik Hvass Pedersen
University of Aarhus, Student #971055
September 2005

1 Introduction

The purpose of this document is to verify attendance of the author to the *Advanced Object Oriented Programming* course, at the Department of Computer Science (DAIMI), University of Aarhus.

A solution is presented for the socalled *Dismiss Problem* in the (graphical) object-oriented programming language called *Self*. The solution is both easy to implement, and solves the Dismiss Problem well.

After an introduction to the Dismiss Problem as well as two seemingly viable solutions, the implementation of the simpler solution is described in more detail. The reader is assumed to be familiar with the project description, as well as the Self programming language, and object-oriented programming in general.

2 The Dismiss Problem

When a popup menu is created in the graphical programming environment for *Self*, the mouse pointer is adjusted so that it is over the menu-item which was last chosen. The exception to this, is if the user had last chosen to *tear off* the popup menu, and drop it down on the background. The button for performing this action, is the top bar having no label.

The *Dismiss* button removes the object in question from the Self GUI, and flags it as deleted in the internal storage system, so that its occupied storage will be reclaimed by the garbage collector. Performing the dismiss operation inadvertently, will therefore have unfortunate consequences, as the work is lost.

This may happen if the user chooses to dismiss one object, and then holds the right mouse-button on another object, but changes her mind about performing an action on that object, and simply releases the mouse-button without having moved the pointer away from the dismiss-button. There is a small time-slice within which the action will be ignored, but after this brief moment has elapsed, the system will register the release of the mouse-button as a request for performing the action in question – here the dismiss operation.

2.1 Yes/No Confirmation

The project description has numerous suggestions to deal with the Dismiss Problem, for example various kinds of trash cans in which objects are placed upon being deleted (dismissed).

Another suggestion from the project description, is to have a confirmation dialogue, whether the user really wants to delete the object or not. This is a good solution because it is simple and fairly common – any text editor for example, has a confirmation of whether or not one wants to discard the changes.

However, there are numerous differences between, say, text-editors and Self objects. First, a text-editor only asks for confirmation if there are unsaved changes, which does not make sense in Self, as changes to an object's slots are saved or discarded in text-editors of their own.

Furthermore, the dismissal of objects seems to be a natural action of programming in the Self GUI, which makes it awkward to keep having to confirm the action.

2.2 Ignore Dismiss Choice

A more graceful but even simpler solution to the Dismiss Problem, is to ignore depressions of the Dismiss button as the last-choice button – the same as for the tear-off button, as mentioned in section 2. This is the approach we use below, and it turns out to work even better than expected.

3 Implementation

Implementation is carried out in Self version 4.2, as installed in the lecturer's directory on the DAIMI computer network, and is made as a socalled *hack* to the supplied `BareBones.snap` snapshot, and saved again as a snapshot in `DismissProblem.snap`.

3.1 Menu Prototype

First we need to find the menu prototype and the code that stores the last-choice button. To do this, hold down the right mouse-button over any object, so as to bring up the menu in question. Select the tear-off button to drop down the menu. Now point with the mouse on this menu, and hold down the right mouse-button once more. This brings up the same menu, but referrering to the object for the menu that we had just tore off. Now select the outliner-button, which brings up the particulars for that object.

This however, is only one such object, and we wish to modify or hack the prototype object for that menu, so that all menus that are either derived or copied from this prototype, will be affected. Therefore we must open the outliner for that object's prototype, which turns out to be called `ui2Menu`.

3.2 Event-Handling Code

After some searching about, trying to understand how the ui2Menu is implemented, we find that we need to modify the leftMouseUp slot under the events grouping. Towards the bottom of that slot, we see the following code:

```
markedButton = tearOffButton ifFalse: [
    defaultButtonHolder rememberButton: markedButton In: self.
    popDown. wrld displayUpdateNow.
    ].
```

This code first checks to see if the object for the button that has been marked, is really the tear-off button, and if not, update the last-choice button by sending a rememberButton message with appropriate parameters to the defaultButtonHolder object. Then the menu is *popped down* (that is, destroyed) after which the display is updated.

3.3 Dismiss Hack

The markedButton object is actually an ui2Button object, which has a label slot used for storing the textual label printed on that button. In the case of the dismiss-button, this string is of course 'Dismiss'. So a simple modification is to check if the label on the markedButton is 'Dismiss', and if not, go ahead and update the last-choice button.

However, earlier in the code for the leftMouseUp slot, various conditions – for example that the user has released to mouse-button right away, meaning that no button should be activated – may lead to markedButton being set to nil (no object), so we need to check for this also. The modification therefore becomes:

```
markedButton = tearOffButton ifFalse: [
    markedButton ifNotNil: [
        markedButton label = 'Dismiss' ifFalse: [
            defaultButtonHolder rememberButton: markedButton In: self.
            ].
        ].
    popDown. wrld displayUpdateNow.
    ].
```

Note the usage of the markedButton's label is enclosed in the condition that the button is *not* the tearOffButton, which means we do not need to worry whether the object for tearOffButton, actually has a label or not. As mentioned above, that button certainly does not display a label, but this does not imply its object has no slot named label, merely holding an empty string. But again, we need not worry about this.

3.3.1 Performance Issues

This solution naturally requires a string compare whenever `markedButton` is not `nil`, regardless of which button was pressed. However, the strings are so short, and the operation is still only performed once per user activation of that menu, so the computational overhead is considered negligible.

3.3.2 Alternative Implementation

Still, one could have chosen other ways to implement this particular solution to the Dismiss Problem, and for various reasons. For example, one could have stored a reference to the object for the dismiss-button similarly to `tearOffButton`, and then use a `markedButton = dismissButton` comparison instead of the above string-comparison. This also has the advantage of not requiring the `markedButton` object to actually have a `label` stored in it (for example, one could use images or symbols instead of labels, and thus reuse the event-handling code without modifications).

However, we will need to make more modifications to the code elsewhere in the `ui2Menu` prototype object, in order to store the reference to the dismiss-button, which in turn means more maintenance difficulties, and the solution from section 3.3 is therefore preferred.

4 Testing

Testing was conducted during the experimental implementation, that is to say, in the process of understanding how the `ui2Menu` object was implemented, a number of different approaches to hacking the code were tried. For example, the circumstances in which `markedButton` was `nil` were explored, and it was found both from manual experimentation and studying the code of the `leftMouseUp` slot, that the hack should be done as in section 3.3 – for which no bugs were uncovered from manual testing, that is, clicking and releasing the right mouse-button, over and away from the menu, and so on.

Testing also proved this solution to the Dismiss Problem to be very smooth in user operation, as the user is always forced to move the mouse pointer over the Dismiss-button and release the right mouse-button, whenever a Dismiss-operation is desired. This means selecting the Dismiss-button becomes a confirmation of the action in itself, and thus alleviates the need for a yes/no dialogue as described in section 2.1.