2.3.2 HE: Error Prevention

In "1.3.1 Basic Heuristic Evaluation," we introduced this heuristic in the following way:

As much as possible, prevent errors from happening in the first place. For instance, if on one application interface there are only a limited number of legal actions, then help users chose from those legal actions. That is, don't allow them to perform any action at all and then tell them afterwards when they've chosen an illegal one. This could be considered a subset of the first heuristic, **visibility of system status** (what input the computer system is ready to accept), but it is so important and so often violated, that it warrants its own heuristic.

Prevent Errors

People explore interfaces and learn by trial and error. As much as possible, disallow actions that are illegal. For example, both MacOS and Windows *dim* (or *gray*) and disable illegal actions. Dimming an action is preferable to removing the action from a menu or dialog box, because when an action is dimmed, its position on the interface menu doesn't change, only its availability. As another example, in a Macintosh file dialog box, only files that an application is capable of opening are listed. The other files in the folder that the application cannot open are not shown, reducing the chance that the user will attempt to open them.

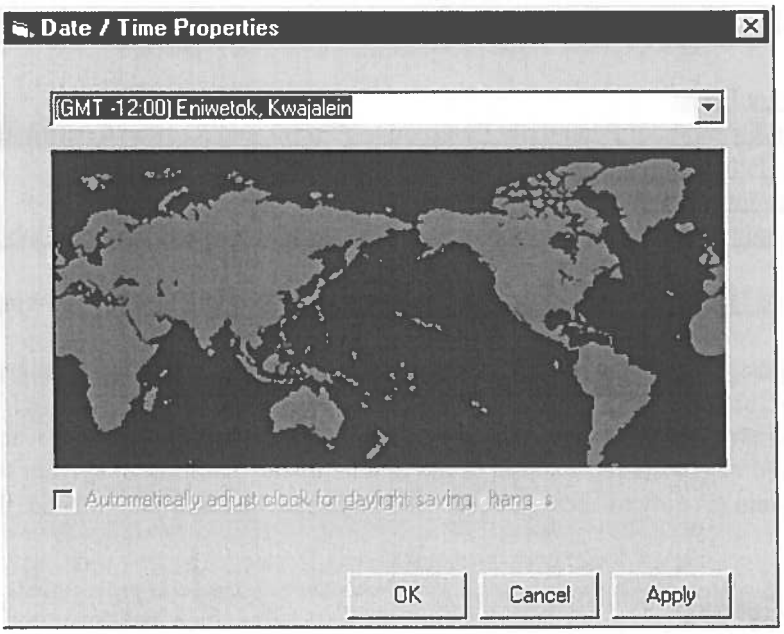
Warn Users about Potentially Destructive Actions and Require Confirmation

Warning messages should be written in the user's language and should provide the user with the option of canceling the command. Avoid double negatives in messages, especially in warnings. For example, the message "If you don't want this to not be saved, click YES" requires careful reading to determine what the consequences of clicking YES would. Make sure warnings clearly state the consequences of continuing with the action. Destructive actions should require explicit confirmation.

A more detailed discussion of this heuristic can be found in Section 5.9 of Nielsen's *Usability Engineering*. Here we give a UAR that applies this heuristic to the design of the Date/Time Control Panel you will implement in Exercise 5.

Date/Time Control Panel Applications of this Heuristic

The following picture shows what your first VB prototype will look like when you have completed Exercise 5. We will examine this small interface using the **error prevention** heuristic and write a UAR for our conclusions.



The result of completing Exercise 5.

When Interface Aspects Are Repeated

This prototype is very similar to the date–setting prototype you created in Exercise 4 and has the same trade–offs in following heuristics for the similar features. For instance, the buttons at the bottom follow the *Windows Design Guide* (UAR# HE5), but the ***OK*** and ***Apply*** labels may be confusing to users (UAR# HE6). The Cancel button provides an emergency exit (UAR# HE7) but it may not provide enough feedback about when it will actually cancel changes (UAR# HE8). Finally, keyboard shortcuts exist for the ***OK*** and ***Apply*** buttons (UAR# HE9).

If you are following a standard, it is not necessary to write up separate UARs for each occurrence of the standard. In a real–world project, you could do just as we did here. Start with a single screen and write separate UARs for each aspect of that screen. When you go to the next screen and discover that the same standards are followed, you can write a set of UARs for the application concerning the standard in general (that is, generalized versions of UARs # HE5–9). And, then write one UAR for the consistency of the system's own interface (because following a standard consistently throughout a whole system is in line with the **consistency and standards** heuristic itself).

Example UAR: Automatic Adjustment of Daylight Savings Time Is Good

UAR Identifier

HE10 — Good Feature

Succinct description:

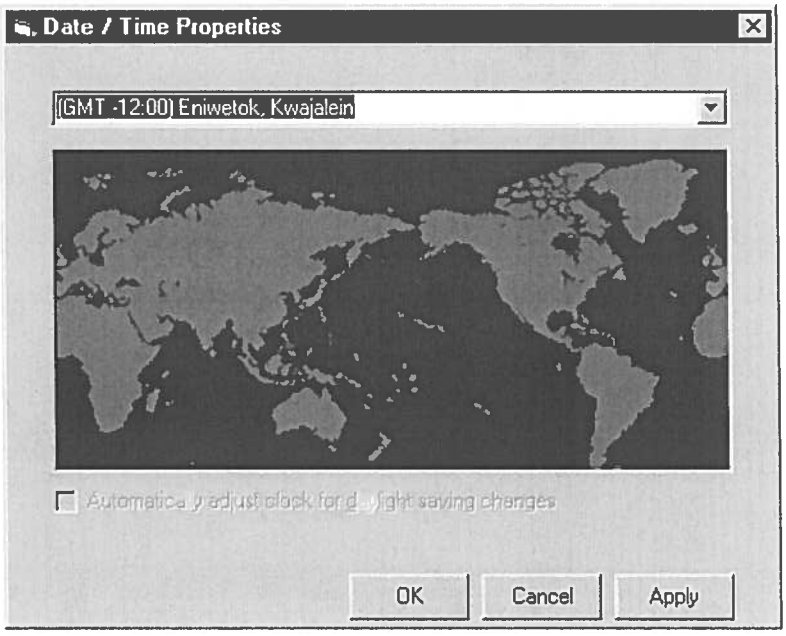
Users need not know Daylight Savings Time rules because system automatically adjusts.

Evidence for the aspect:

Heuristic: error prevention

Interface aspect:

A CheckBox (under the map in the figure below) allows the user to instruct the computer to automatically track Daylight Savings Time (DST) for the selected time zone and adjust the system clock at the appropriate time and day.



Explanation of the aspect:

Being able to have the computer automatically track the rules about daylight savings time is very good, because people are likely to make errors. Different geographic and political areas in the world use different rules for DST, and no one person is likely to know all of them, but they can be programmed into the computer. For instance, I was one hour late for a conference call to England this spring because they changed to DST a week before the U.S. did and I had no knowledge of this cultural difference.

There is nothing wrong with putting a personal anecdote into a UAR if it serves to make an abstract point concrete.

Benefit of the good feature:

The users will not have to know about all the special rules about changing to Daylight Savings Time. This will prevent errors for all types of users (novice, occasional, and skilled).

Solution:

I cannot foresee any problems with this implementation, as long as the computer's information about Daylight Savings Time is correct. We will have to find a definitive source of this information (some International Bureau of Time?) to make sure the system is correct.

Relationship to other UARs:

None when the original UAR was written.

The course provides an appendix of a complete set of UARs that analyze the Date/Time control panel, and you can take this link directly to "Appendix A. Date/Time Control Panel UARs."
