WSE: An Environment for Exploring Window Strategies

Marja-Riitta Koivunen

Department of Computer Science†
Human Computer Interaction Laboratory
University of Maryland

This paper describes Window Strategies Environment (WSE), which is an environment for exploring different window strategies. It is a tool for developers who want to implement and explore different window strategies easily. WSE uses object oriented methodology offering four hierarchical levels for customizing predefined window strategies. The highest level is a simple modifiable language for describing a few essential commands for each strategy.

This kind of tool helps the developers to implement and test several approaches to window strategies before selecting one. Also it encourages the developer to direct some of the power to the user so that she can customize the user interface by selecting a preferred strategy from a predefined set of strategies.

The environment is implemented under NeWS 1.1 [Sun87] window system for SUN 4/110. It uses Smalltalk style object oriented class mechanisms [Goldberg83, Densmore] offered by NeWS but none of the original NeWS toolkits.

1. Introduction

A messy desktop on the screen has been a common phenomenon after discovering the benefits of multiwindow user interfaces [Shneiderman87]. Since then the users have been living with a dilemma of wanting some typical opening, closing, resizing and moving routines to be automated, but not in a way that unpredictably loses the information into neat looking but confusing piles.

A typical set of routines needed to organize windows after selecting an application operation is referred to as a window strategy. Often window strategies are divided into two categories

† Current address: Department of Computer Science
Laboratory of Information Processing Science
Helsinki University of Technology
according to the possible placements of the windows they are using: tiled or overlapped windows. Usually, however, total overlapping does not make much sense in a strategy. At least a small handle to a window is needed so that the user is able to utilize it.

Automatic window strategies can be very powerful in supporting users' normal routines. For instance, Smalltalk browser [Goldberg84], as well as, the NeXT machine user interface [Thompson88] successfully apply some simple window strategies. The NeXT directory browser chains several windows together horizontally. A selection of a file or a directory in one window creates as a result the contents of the selected object. This result is then shown in a window adjacent to the manipulated window. This kind of strategy emphasizes the concept of the directory hierarchy as it always shows part of the directory path. The power of a strategy naturally depends not only on the application but also on the user. For instance, one user may not want the contents of an adjacent window to disappear.

There does not seem to be any simple universal solution to the messy desktop dilemma. The more automatic changes happen on the screen, the easier it is for the users to get distracted from their tasks [Lifshitz89]. Fortunately, partial customized solutions can be found if enough time and effort is put into developing them. However, the current window toolkits require too much effort to generate several different approaches for testing purposes. Often only one strategy is developed or the organizing of the windows is left totally for the user.

As the Human Computer Interaction Laboratory in University of Maryland had a strong need for developing window strategies for hypertext applications [Hyperties89, Lifshitz89, Seabrook89, Shneiderman86], an environment for easing the development was designed. The Window Strategies Environment as such is not restricted to be used only with hypertext, merely it is a good practical application area. Selecting NeWS [Sun87] as the underlying window environment, however, was enforced by the currently developed hypertext system for SUN, Hyperties [Koivunen89].

2. Sample Window Strategies

Some simplified window strategies are mentioned as an example when the architecture of the WSE is described. These strategies are by no means meant to explore all the capabilities of the environment, merely they try to illustrate the main ideas behind its structure. All the sample strategies use tiled windows, and can be easily applied to hypertext systems. Careful user testing is naturally needed to evaluate their usefulness.

To ease the definition of the strategies, two concepts are defined. A window that is manipulated by the user to create a new result to be shown to the user is called a source window. Correspondingly, a window where the created result is shown is called a destination window. These names can naturally also refer to the same physical window.

2.1. Chain Strategy

Chain strategy is a very simple strategy. Basically it means that a destination window is attached to every source window. When the contents of the source window are manipulated the destination window attached to it will be used to show the results. If the destination
window is the same as the source window the original contents of the source window will
disappear after the manipulation.

![Diagram](image)

Figure 2-1: Sample Chain Strategies.

Here is a figure with two typical chain strategies. The (a) strategy has three windows, which
will all show the results in a window to the right of them. The last window in the chain
points to itself. The (b) strategy consists of four windows, which all point to the next
window clockwise from their location.

2.2. User Drag Strategy

This strategy gives more power to the user in arranging articles, program code etc. to
windows than the chain strategy. After manipulating the source window the user selects the
destination window explicitly. The selection can be done, for instance, by moving the cursor
to the destination window. This strategy is good in a sense that information is not lost
automatically, the user always does it herself. On the other hand this approach requires more
participation from the user, which can be disturbing especially with routine tasks.

Figure 2-2 illustrates one user drag strategy. There are five snapshots of the user interface,
that are taken just after the mouse button under each window was in the given state. The
mouse in between snapshots describes the intervening cursor movement. At first an item is
selected from the contents of the source window by pressing a mouse button down while the
cursor is above the item. The destination window is then defined by letting the user drag the
cursor to one of the available windows while keeping the mouse button down. When the
button is released the last visited window is used as the destination window. If no dragging
is done the contents of the source window is replaced.
2.3. Domain Specific Strategies

None of the earlier defined strategies take the domain hierarchy into account. For instance, in some applications the figure 2-1 (a) strategy could always show the last selection at the rightmost window and the selections preceding it in timely order in the two leftmost windows. However, the goodness of these kinds of approaches depends highly on the domain and are not discussed further here.
3. Architecture

The main aim of the WSE is to offer two important but partially controversial qualities:
1. The environment is simple to understand and use.
2. Most aspects of the environment are easily customized.

Both of the demands above are met by giving the developers several different levels of customization. The higher the customization level is the simpler it is to use. However, even the system dependent levels can be easily changed without forcing the developer to write large parts of the code all over again. An Smalltalk style object oriented methodology [Goldberg83] was chosen as a tool for implementing these levels since it supports fast and effective customization.

3.1. Levels of Customization

The classes defined in WSE are roughly divided into four customization levels: system dependent level, component level, strategy level, and command level (see figure 3-1 (a)).

![Diagram of Levels of Customization]

Figure 3-1: Levels of customization.

The normal operating level is the command level, which means that the developer uses simple high level language commands specialized for her problem area. In this case a rich set of classes suitable for that problem area are usually defined ahead of time. For instance, when window strategies are developed for hypertext applications, a rich set of hypertext strategies offer enough variety to the developer to stay in this level.
The system dependent level is the lowest customization level. It uses the possibilities offered by the underlying system directly. These possibilities are then gathered to several classes offering a system independent interface to the next level components. For instance, the coding of device dependent input operations is done in this level.

The component level defines specially tailored versions of the system dependent level classes. The component classes can send messages specific to the classes that are going to use them. So the tailoring meets the needs of one or several application classes, such as, a certain strategy class or all the window strategy classes.

Finally the strategy level is a level where different components are put together to form a "touch and feel" to a class object. At this level the main issues are what components are needed and how they will function together. Some strategies, such as, button strategies may take an extra definition cycle and form button component classes, which can be used by window strategies or some other application classes (see figure 3-1 (b)).

3.2. Main Object Hierarchy

The decision to use an object oriented methodology as a tool for developing the WSE gives a great starting point. However, a good foundation for the environment has yet to be established. The most important decision to make is what kind of classes are needed and what is the best way to categorize them and place them into an inheritance tree.

Figure 3-2 shows the main class categories and their usual inheritance order in the WSE. The root class Object is divided into three main class categories: Appearance classes, Behavior Classes, and Property classes. Property classes are general classes that can be fused into different parts of the tree by using multiple inheritance.

All the classes in Strategy category are subclasses of an appearance class. Behavior classes on the other hand are used by the Strategy classes to create different "touches and feels". This organization makes sense, because the appearance methods are more likely to be needed by behavior methods than vice versa in user driven interfaces. All the categories are discussed in more detail in the following sections.

![Diagram of Main Class Categories](image-url)
3.3. Appearance and Behavior in a Strategy

A window strategy clearly has two main characteristics: appearance and behavior. The division into these two classes is not, however, as simple as it might look. The behavior and appearance are often closely intertwined together in a strategy. For instance, in user drag strategy the window where the cursor is dragged to, is highlighted. This makes it difficult to obtain a clear separation. Here the division is made as follows.

The behavior classes define all system dependent interaction used by the strategies. A behavior class is then used as a parent class to several subclasses. Every subclass creates a special behavior component for a certain strategy or some strategies. Figure 3-3 illustrates the tree structure. The component classes are underlined.

A strategy class inherits its basic appearance from an appearance class and selects a behavior class component tailored to its needs. All the special features of a certain strategy are defined in the strategy class. This kind of basic structure makes it easy to understand the strategies, without having to browse through a lot of code.
The button strategies (figure 3-4) are an example of component strategies. They are used by window strategies. Button strategies use a special appearance subclass and they are usually handled as button sets rather than as individual buttons. Button strategies are always used by a window strategy and they can be specialized to meet the needs of a certain window strategy in the same way as the behavior classes can be specialized.

3.4. Properties

The classes in an inheritance tree often utilize similar parts of autonomous code. These classes are referred to as host classes. These similar code parts are often hard to generalize in a tree. The generalization is especially difficult if the aim is to keep the class definitions as simple as possible. Too much generalization in a very heterogeneous tree decreases the understandability of the tree structure. On the other hand writing the same code repeatedly to host classes located in different branches of the tree is not only error prone but also unfriendly to the developer. Sometimes it is possible to write independent classes containing the similar code parts, and utilize them in host classes. However, also this can lead to a lot of copying, because often the method names are needed to be available in the host class domain as well.

In WSE the autonomous code parts were written as separate classes called properties and handled by using multiple inheritance. Since the class model in NeWS did not have multiple
inheritance it was simulated by copying the properties before the host class to the inheritance
tree. This did not totally alleviate the need to copy, but it did increase the simplicity of classes
and make the tree structure easier to understand. Because the properties were inserted into the
tree just before the host classes, the methods of the properties dominated the more general
methods in the tree with the same name. So special care was taken not to create unwanted
name conflicts.

The interface between the host class and the properties can be a two way interface. It means
that in addition to the property methods that are used by the host classes, the property can
also use some methods of the host class. However, usually the interface from a property to a
host is as minimal as possible. Otherwise the usefulness of a property becomes too limited.

The usage of the properties are illustrated in figure 3-5. The properties are in parentheses.
The figure describes the inheritance tree of a few windows strategies. In some strategies,
such as the earlier described ChainStrategy, the windows may be self-contained. It means
that they are not related to other windows. Other strategies consist of windows which belong
to window groups, or windows which have child windows. In a tree of windows, a window
belongs to a window group as well as has child windows. All these strategies often use the
same kinds of methods to define group and child window relations. For instance, a group
window must be able to add and delete its children and perform methods to all of them.
These sets of methods are provided by properties named ChildProperty and GroupProperty.
By using these properties it is easy to change, for instance, a simple window to a tree of
windows without having to write much code.

![Diagram of WindowStrategy class hierarchy]

Figure 3-5: Class hierarchy of WindowStrategy category.

3.5. Application Interface

Usually the main part of the application interface is defined in the strategy level. This helps to
isolate the interface from the rest of the environment. If an interface has a very complicated
behavior or it needs some special system dependent features, it may need to customize some
classes in the component or system dependent levels too. However, in this case the classes are also changed in a controlled manner. Otherwise the application interface is not restricted to any particular form. Different approaches can be taken in different strategies.

4. Future Research

As WSE is quite new, only standalone window strategies targeted for hypertext applications have been developed. These strategies function as if selections are made from normal hypertext articles, but currently the articles are actually static definitions. Most of the strategies are aimed to be used in the future by Hyperties. This means that in the future the strategies should be easy to define by using WSE, and to integrate into Hyperties as new user interfaces. This will encourage a lot more testing of the user interfaces as the interfaces are easier to create and modify. Automatic gathering for information and other support for the evaluation purposes would increase the usability even more.

Figure 4-1: A simple screen organization script.

Currently the screen organization is usually defined first and then used by a selected strategy. In the future screen organization scripts, such as in figure 4-1, could also be used. The scripts define how the user interface changes over time according to the selections made from certain windows. However, the changes are predictable as they always follow the guidelines defined by the scripts.

Another important goal in the future is to develop a graphical user interface to the system, for instance, akin to Brad Myers Peridot [Myers88]. This will offer one more level of customization, namely a graphical level. In this level the windows, groups, and buttons will be defined graphically by using direct manipulation. For instance, a window could be create
by drawing a rectangular area on the screen. After that a name field of that window will be filled-in. Buttons attached with rubber-links will be created to one window and then dragged to another window to define a hypertext link between the windows. The user interface will then be stored to a file and used, for instance, as a style file for Hyperties.

5. Conclusions

To give more power to the user does not mean that we have to burden her with unnecessary tasks. Organizing and sizing of the windows could be often handled automatically if enough user testing and design effort are put to the development. As the currently available window toolkits or UIMS offer little help in developing window strategies, it is convenient to leave most of these tasks to a user, or test only a few alternative strategies. Clearly a specially tailored window strategies environment is needed.

The current experience of the developed environment is positive. It is easy to understand, use, and modify. Four available customization levels add clarity and structure to the system without limiting its flexibility. Several window strategies have been implemented by using it. Some of them are strategies that simulate already existing strategies used by some experimental Hyperties versions, and some are totally new designs. However, WSE needs to be developed further to offer a richer set of alternatives. Also, as more strategies are developed in the future we will have more experience to evaluate the environment properly.

NeWS was selected for the implementation because the new Hyperties version was already using it. NeWS offered a very flexible and rich environment for development. Especially the available graphics operations were remarkable. However, good programming tools, such as, class browsers or debuggers were not available at the time being. A better class model would also have helped a lot. Especially in a very general system such as WSE, multiple inheritance would have been really useful.

Acknowledgements

I want to thank prof. Ben Shneiderman for his comments and support during the development of this work. Don Hopkins' CyberSpace NeWS debugger, which he was developing at the time, helped me enormously when I was struggling with NeWS. Finally thanks to the rest of our SUN Hyperties team: Catherine Plaisant-Schwann, Rodrigo Botafogo, and William Weiland. It was a pleasure to work with them.

References

[Densmore86]
Densmore, O. (1986)

[Goldberg83]
Smalltalk-80: The Language and its Implementation. Addison-Wesley, p. 714.
[Goldberg84]
Smalltalk-80: The Interactive Programming Environment. Addison-Wesley, p. 516.

[Hyperties89]
Cognetics Corp. (1989)

[Koivunen89]

[Lifshitz89]
Window Control Strategies for On-Line Text Traversal. M.Sc. Theses, Department of Computer Science, University of Maryland.

[Myers88]
Myers, B. (1988)

[Seabrook89]
The User Interface in a Hypertext, Multi-window Program Browser. Interacting with Computers, Vol. 1, No. 3.

[Shneiderman86]

[Shneiderman87]
Shneiderman, B. (1987)
Designing the User Interface. Addison-Wesley, p. 448.

[Sun87]
Sun Microsystems, Inc. (1987)

[Thompson88]
Thompson, T. and Baran, N. (1988)