A Constraint Interface for Managing Windows

Greg J. Badros Jeffrey W. Nichols Alan Borning
{gjb,jwnichls,borning}@cs.washington.edu
Dept. of Computer Science and Engineering
University of Washington, Box 352350
Seattle, WA 98195-2350 USA

ABSTRACT
Constraints are an important user interface technology. We developed SCWM as a testbed for exploring UIs for constraint-based layout. Its constraint interface includes a toolbar for adding relationships and an "investigator" for viewing and manipulating the constraints. We performed a discount usability study with six users and learned that our interface was sufficient for them to complete the tasks, but also uncovered numerous usability issues. We have improved our interface from that feedback and suggest more studies testing our other interaction paradigms including constraint inference and voice-recognition.

Keywords
Window manager, constraints, layout, Scheme.

INTRODUCTION
Constraints are an important user interface technology. They permit users to specify relationships that they desire to hold among entities. Constraint-solving algorithms then maintain these relationships automatically. Techniques for interactively specifying and manipulating constraints are not well-established. We developed SCWM—the Scheme Constraints Window Manager—as a testbed for exploring UIs for constraint-based layout [2]. The constraint features enable users to better integrate related application windows. For example, a user may desire an `xterm` and an `emacs` that are running on the same machine to move around together, or for a window displaying the current time to stay visible in the bottom-right corner of the screen.

To maintain constraints among application windows, SCWM embeds our Cassowary Constraint solving toolkit (software and papers are available from `www.cs.washington.edu/research/constraints/cassowary`). SCWM is a practical system that leverages Scheme to simplify prototyping advanced functionality and has hundreds of users in the Unix X-Windows community.

CONSTRAINTS INTERFACE
Our user interface consists of two main parts: a toolbar for adding relationships (fig. 1) and an investigator utility for examining the current constraints (fig. 2).

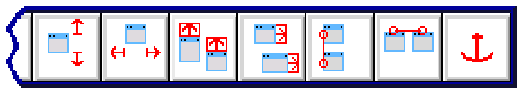
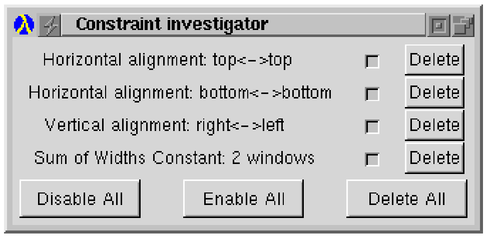


Figure 1: The right half of the constraint toolbar.

Each toolbar button has an icon that represents a kind of constraint and provides pop-up help to aid new users in learning the represented relationship. To add a constraint, users click on a button and then select windows to relate. Some constraints affect only a single window (e.g., the rightmost button anchors a window) and others relate two or more windows (e.g., the second button from the right initiates horizontal-alignment).

For many of the constraints, further parameters of the relationship are determined by *where* users click when selecting a window—e.g., an anchor keeps a window's northwest corner in place if users click in the upper-leftmost ninth of a window, but it instead keeps the southeast corner fixed if they pick the bottom-rightmost ninth. This behaviour is suggested by highlighting an edge or corner of the window being selected.


Constraint investigator

Horizontal alignment: top<–>top	☐	Delete
Horizontal alignment: bottom<–>bottom	☐	Delete
Vertical alignment: right<–>left	☐	Delete
Sum of Widths Constant: 2 windows	☐	Delete

Disable All Enable All Delete All

Figure 2: The constraint investigator window.

The constraint investigator window lists all constraints added by the user. Each constraint may be enabled/disabled via a check box or deleted by clicking a button. Parameters of each constraint are listed and moving the pointer over the checkbox highlights the related windows and marks-up those actual windows with a visual representation of that constraint. When constraints are re-enabled, the affected windows animate to their new positions.

USABILITY STUDY

We applied a discount usability approach [4] to improve our constraint interface to managing windows.

Methodology

Six advanced computer users thought aloud while performing three tasks. Each task consists of two parts: discovery and re-creation. First, users manipulate windows with constraints already active to discover and describe those relationships (without use of the constraint investigator). After giving a correct description, they then use the interface on a second display to constrain a fresh set of windows identically. Users were given only a very minimal description of the interface.

The three constraint configurations tested were: 1) a Netscape Find dialog kept in the upper right corner of the main browser window; 2) three windows kept right-aligned along the edge of the screen such that none of the windows overlap nor leaves the top or bottom of the screen; and 3) two windows tiled horizontally.

Results

All users were able to complete their tasks. Discovering the constraints was straightforward—manipulating the windows and observing the behaviour was sufficient to deduce the relationships already present. Re-creating the configurations was more troublesome, but users still succeeded. They often used the investigator to remove incorrect constraints, but then continued onward with an alternate hypothesis.

Problems discovered

Our study uncovered numerous usability issues. The most substantial problem involved selecting window parts for the alignment constraints. When performing a vertical alignment, all that matters is whether the user clicks on the left, center, or right third of the window—it is irrelevant whether the click is in the top, middle, or bottom of the window. Our interface, however, still highlighted individual corners or edges as it does for anchor constraints where any of the nine positions is significant. Users were confused by the UI distinguishing along the irrelevant vertical dimension. We revised Scwm to highlight whole edges of windows when applying an alignment constraint.

When users began adding a constraint and wanted to cancel, they were unsure of how to abort their action. Some users clicked on the toolbar thinking that is a special window. Others discovered that clicking on the background results in an error that terminates the operation. No user realized that a right-click aborts and we now also support pressing the `Escape` key to cancel a window selection.

Other observations

The users who performed best studied the tooltip help for each of the toolbar buttons before attempting their first re-creation sub-task. We were surprised at the variety of constraints used in re-creating our configurations: no user matched the expected solution on all three tasks. In particular, the "strict relative position" constraint was used especially advantageously by users who chose to configure windows manually before applying constraints to keep the windows as they were.

Not all users discovered the constraint-visualization feature of the investigator. We now draw the visualizations whenever the user points at any part of the description, not just the enable checkbox. Also, one user wanted to directly modify the parameters of a constraint in the investigator window.

Users definitely enjoyed the constraint features. One commented: "Wheee! They bumped into each other—that was pretty cool!"

CONCLUSIONS AND FUTURE WORK

Our constraint interface was usable for realistic tasks. Users can understand constrained layout on the basis of the dynamic behaviour of windows and can use our toolbar and investigator to re-create the constraints anew without any training in the use of the UI.

Managing windows provides a highly-dynamic environment to experiment with user understanding of constraints. Lessons learned from Scwm are applicable to more static layout domains such as web page layout [1].

Although the interface to the constraints tested here is fairly conventional, Scwm supports both constraint inference [3] and voice recognition for establishing and manipulating relationships on windows. We also provide a programming-by-demonstration facility to create abstractions of compositions of constraints. These interaction paradigms may lead to substantially better user experiences. We will perform more usability studies to refine and improve those techniques and compare them to this initial interface.

ACKNOWLEDGEMENTS

We thank all those who participated in our study. This research is funded by NSF Grant No. IIS-9975990 and the University of Washington CSE Wilma Bradley fellowship for the first author.

REFERENCES

[1] G. J. Badros, A. Borning, K. Marriott, and P. Stuckey. Constraint cascading style sheets for the web. In *Proceedings of the 1999 ACM Conference on User Interface Software and Technology*, November 1999.

[2] G. J. Badros, J. Nichols, and A. Borning. Scwm—the Scheme Constraints Window Manager. In *Proceedings of the AAAI Spring Symposium on Smart Graphics*, March 2000. To appear.

[3] B. A. Myers and W. Buxton. Creating highly-interactive and graphical user interfaces by demonstration. In *Proceedings of SIGGRAPH 1986*, Dallas, August 1986.

[4] J. Nielson. *Usability Engineering.* Morgan Kaufmann, 1994.