Grouping and Ordering User Interface Components

Mark H. Gray
College of Computing, Georgia Institute of Technology,
Atlanta, GA 30332-0280, *email:* mark.gray@cc.gatech.edu

James D. Foley
College of Computing, Georgia Institute of Technology,
Atlanta, GA 30332-0280, *email:* foley@cc.gatech.edu

Kevin E. Mullet
Human Interface Technology Group, SunSoft Inc.,
2550 Garcia Ave. MS 1-07, Mountain View, CA 94043-1100, *email:* kevin.mullet@sun.eng.com

ABSTRACT

In automatically generating a user interface from a model of the target application, many factors that affect the resulting interface's quality must be considered. Any available semantic information that can improve the interface should be used. Application actions or action parameters may be related in ways that affect placement of their associated controls in dialogue boxes. Two relationships considered here are grouping and ordering. Grouped objects should appear together, possibly visually separated from other controls, and controls which have a logical sequential ordering should appear in that order. We present an algorithm for creating an ordering of controls which correctly satisfies these constraints.

KEYWORDS: User Interface Software, Automatic User Interface Design, Data Models, Dependencies, Grouping, Ordering

1 INTRODUCTION

Automatically generating a user interface from an application model can speed the user interface development process. Using grouping and ordering information can allow generation of better interfaces. For this to be possible, grouping information must be either explicitly or implicitly included in the model. The model designer may explicitly group application objects by giving a group name to a collection of objects or implicit grouping information may be extracted from the hierarchy of the data model.

Interface controls frequently have a logical order which comes from a logical order between the underlying application actions. For example, when printing a document there is usually a choice between printing directly to the printer or to a temporary disk file, and there is a control that specifies which file name to print to. It makes logical sense to choose whether to print to a file before providing a file name, so the model designer should create a logical order between these two controls. Similarly, when doing a textual search and replace, most people expect to specify what to search for first, then what to replace it with. This order supports searches without replacement and makes verification of replacement easier.

Consider how much better the layout in figure 1 looks using grouping and ordering information than the randomly ordered layout in figure 2.

2 THE GRAPH ALGORITHM

Maintaining hierarchical grouping and logical ordering simultaneously is a nontrivial task when ordering may occur both within and across groups and an ordering dependency may occur between objects at different levels in the

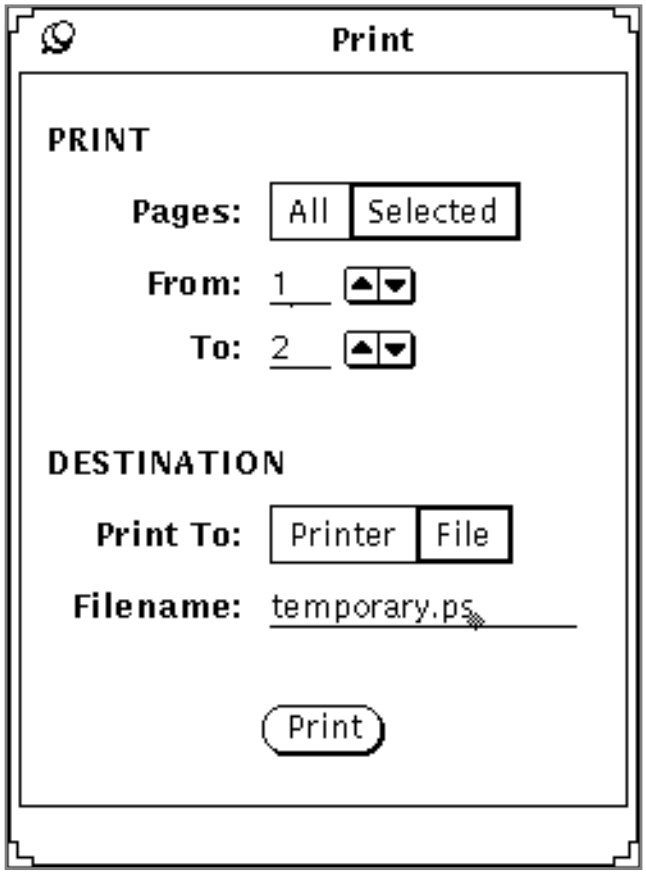

Figure 1: Layout using grouping and ordering information

grouping hierarchy. The current algorithm is described below with pseudocode following.

2.1 The Grouping Hierarchy

Interface objects are organized in a tree structure whose internal nodes are groups and whose leaves are controls. The generated interface will have a one-column layout where controls will be correctly ordered by a simple traversal of this tree once it has been reordered by the algorithm presented in this paper. Each window contains the root of the tree of objects in that window. Siblings in a tree are logically in the same group. Logical ordering of controls is specified by general preconditions and postconditions which may be attached to objects in the application model. If one action has a postcondition (something which becomes true after the action is performed) and a second action has a matching precondition (something which must be true before the action can occur) then a logical ordering constrains the first

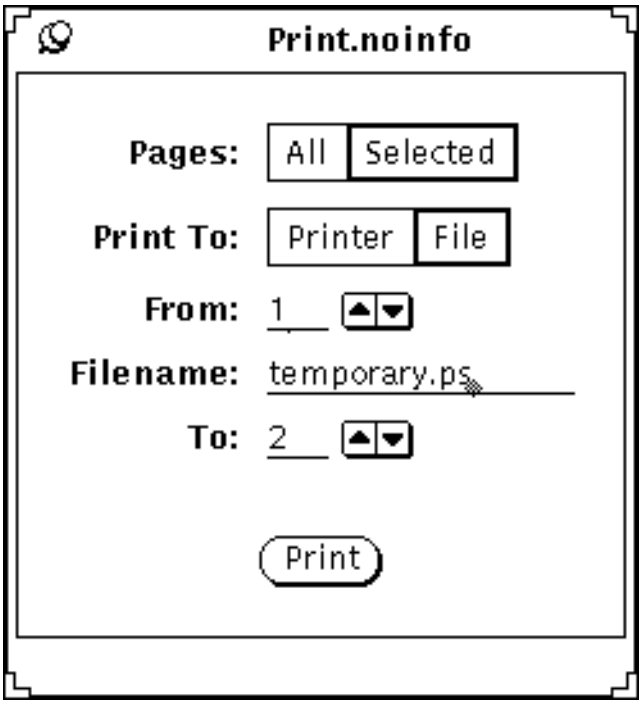

Figure 2: Layout not using grouping or ordering information

action to come before the second. This ordering is recorded in the grouping hierarchy as an additional non-tree edge from one interface object to the other.

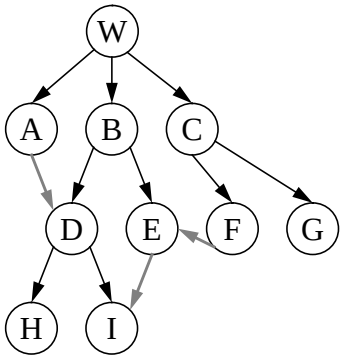

Figure 3: Input graph after insertion
Solid arrows indicate hierarchical grouping
Gray arrows indicate ordering relationships

In the diagram above, black arrows are tree edges which go from a group to members of that group and gray arrows are ordering relationships. Node W is the window whose controls are currently being arranged, so it is a group containing all the other objects. Node A is a leaf node, so it is a control

in the interface. The gray ordering arrow from A to D indicates that A should appear before D in the final interface. Nodes B and C are groups within the window and node D is a subgroup of B.

2.2 Insertion

Interfaces are generated one window at a time. During the generation of a window, controls are individually generated and inserted in the window's grouping hierarchy. New objects like A which belong to no group smaller than the entire window are left at the top level of the hierarchy while those belonging to a group are inserted in their group. If a group or subgroup is not yet represented in the hierarchy, it is created when one of its members is inserted. After all objects are inserted, the grouping hierarchy contains all the intended groups and subgroups of controls, but ordering of the tree does not yet follow any logical ordering constraints given by the data model.

When an object is inserted in the grouping hierarchy, it is also checked against all other controls in the window to determine whether any logical ordering exists between the new control and those already present. If an ordering dependency is found, it is recorded, but the hierarchy is not reorganized to comply with these ordering constraints until all controls have been added. Delaying reordering of the tree means this potentially time-consuming step is performed only once each time controls are added.

2.3 Dependency Propagation

The first step of putting the tree in a logical order is propagating all ordering dependencies between objects that are not siblings up to their ancestors that are. When those ancestors are later placed in logical order, the order between the objects originating the ordering will be preserved by the depth-first nature of a preorder traversal of the resulting tree. Using the above graph as an example, the groups B and C are siblings which contain the objects E and F respectively, and the logical ordering "F precedes E" (denoted F > E) exists, so that dependency is propagated to the parent groups to become C > B. Now a depth-first traversal will visit F before E after the tree is reordered so C does precede B. If dependencies exist in both directions between two groups it is not possible to both keep groups together and satisfy the given orderings. Such a problem would occur in the above example if D or E were constrained to precede F or G. This type of problem as well as simple loops of ordering dependencies are detected and warned about in the next stage of this process. In addition to propagating F > E up to C > B, the example requires two other dependency propagations: A > D to A > B and E > I to E > D. These new edges are added below.

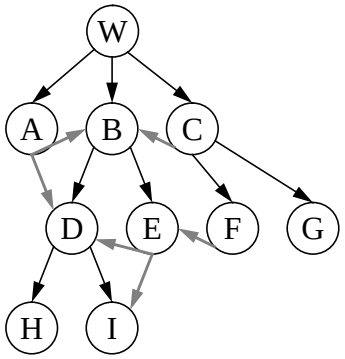

Figure 4: Graph after propagating dependencies up to sibling ancestors

2.4 Ordering Siblings

Once all groups contain all the information necessary to create a correct logical order among siblings, the last step of the algorithm is sorting siblings to satisfy all sibling-sibling ordering arrows. The tree is in a logically correct order when all ordering arrows point from left to right, indicating that the relationships are satisfied. Other ordering information that is not between siblings is redundant and is not shown after figure 4. Each group of siblings is separately sorted into a correct logical order by building a chain of siblings whose links are ordering relationships. If a chain includes all siblings in a group exactly once, that chain represents the only possible logical order of that group. If a chain contains fewer elements than the group, other chains are created starting with unattached members of the group until the collection of chains covers the group. These chains

are then laid out end-to-end to form a correct logical order for the group. If a chain tries to include the same object more than once, either dependencies are present in both directions between groups in the data model as discussed above or a loop of dependencies exists like A > B and B > C and C > A. In either case, it is not possible to satisfy all the given restrictions so a warning is given that at least one constraint is not satisfied. Logical problems like this should not be common, but since they may occur this system must detect them.

In two steps the siblings in the example tree will be sorted. First, siblings E and D under B form the chain E, D so they are placed in that order in the tree. (figure 5).

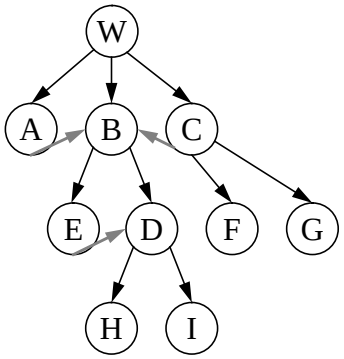

Figure 5: Graph after ordering E and D

Second, siblings A, B and C form the chain A, C, B and they are placed in that order in the tree. (figure 6)

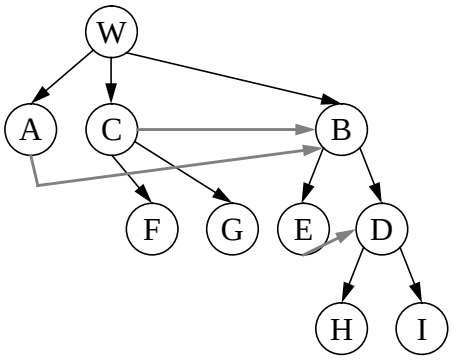

Figure 6: Graph after ordering A, C, and B

After this step, all siblings are sorted so all ordering arrows point from left to right through the tree.

Algorithm Pseudocode

Insertion:

```
for each new interface object
    if object has no grouping information,
        append object to list of top level objects.
    else if object belongs in a group that exists,
        insert it in that group or recursively in a
            subgroup.
    else object belongs in a group that doesn't exist
        so create that group and insert object in it.
    if object has any preconditions or postconditions,
        insert directed logical ordering edges to or from
            objects already present with corresponding
            pre- or postconditions.
```

Dependency Propagation:

```
for every group at every level
    for every pair of objects in that group
        if one object is reachable from the other by
            following any combination of directed
            ordering edges or tree edges,
            insert a new ordering edge between this
                pair of objects.
```

Ordering Siblings:

```
for every group at every level
    while there are objects in the group not in a chain,
        create a chain of objects not already chained by
            following the logical ordering relationships
            between siblings.
        if a chain contains the same object more than
            once,
            break the loop in the chain and warn the
                designer of the illogical input.
        order the group from left to right in chain order
```

2.5 Layout

Controls are placed in their window in a preorder traversal of the grouping hierarchy which both preserves the left-to-right order created by logical orderings and keeps groups together. Layout of groups of controls depends on what interface style is followed. Currently the OPEN LOOK style guide is followed which specifies that groups be visually separated by a small amount of space and an optional bold and fully capitalized group name. Using another style, groups might be shown by a thin bounding box or indentation. Figure 1 in the introduction is created from the group-

ing hierarchy of a simple print dialog pictured in figure 7.

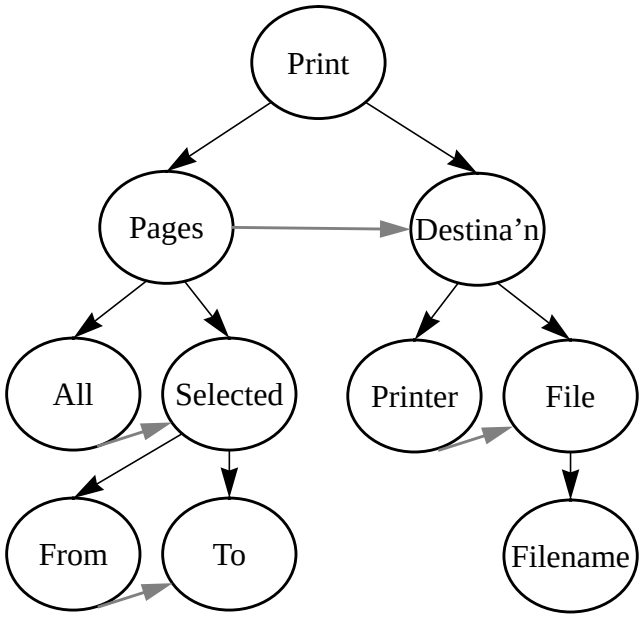

Figure 7: Graph of the print command data model used to create figures 1 and 8

When "Print All Pages" or "Print to Printer" are selected, controls with associated preconditions (and resulting ordering relationships) become disabled as illustrated in figure 8.

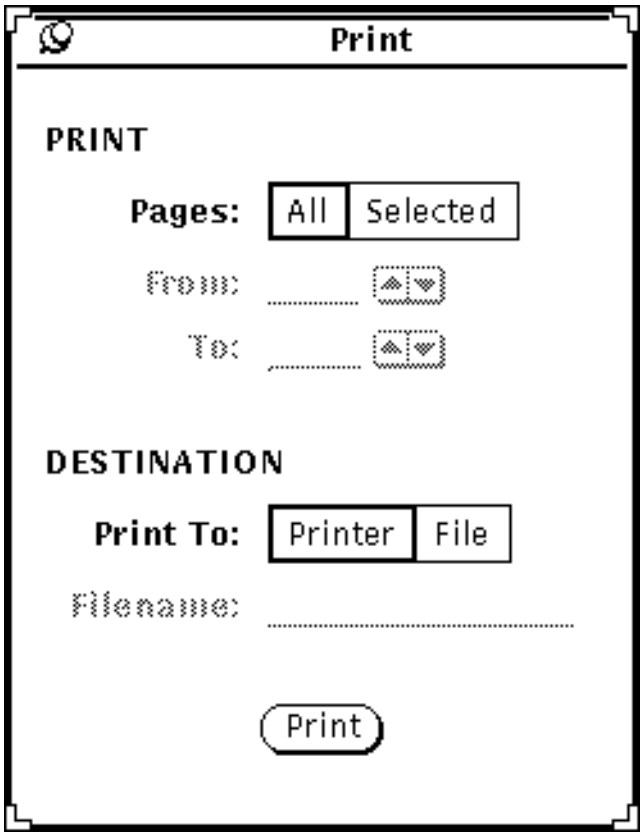

Figure 8: Layout using precondition information both for ordering controls and for disabling inappropriate controls

3. CONCLUSION

Grouping and ordering dependencies have always existed among user interface components, but usually they are not represented explicitly by design tools during user interface design. Instead, if a good designer is building the interface, that person is manually laying out the controls. While placing controls, the designer notices the most obvious possible groups of controls and places them together, then notices the most obvious logical orderings and moves controls to satisfy the perceived orderings.

If the application designer builds an application data model that could be used as input to an automatic user interface generator, that model should contain the desired grouping and ordering information. A good user interface generator can use these relationships to make much better decisions when placing controls in the interface. Hopefully, interfaces whose generation uses this additional semantic information are easier to use because logical order and intuitive grouping from the mind of the application designer is carried through to the eyes of the end user.

ACKNOWLEDGMENTS

We thank Amihood Amir, H. Venkateswaran and James Burns of Georgia Tech for their algorithmic assistance. Funding for this project was provided by SunSoft's Collaborative Research program.