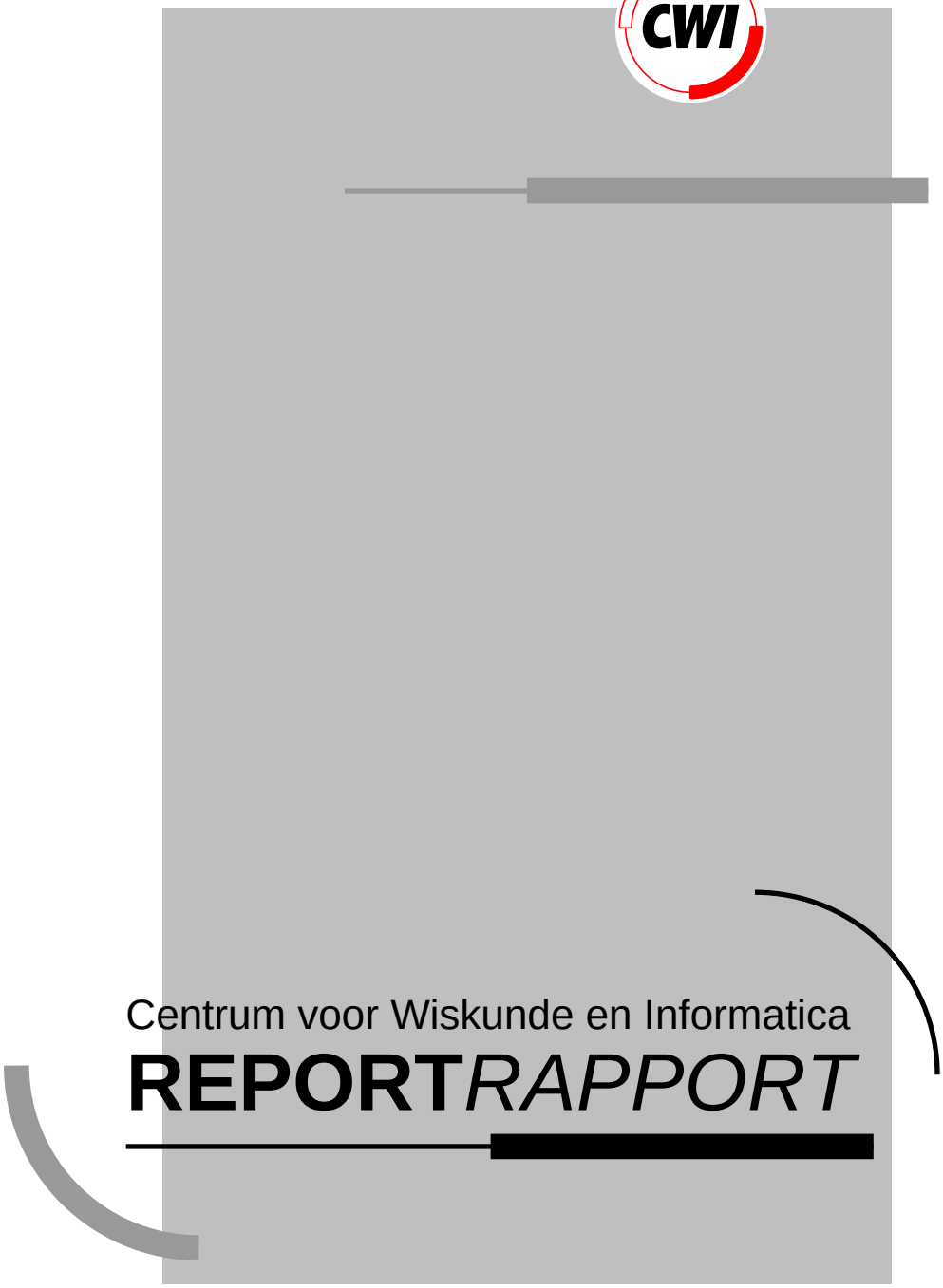

Centrum voor Wiskunde en Informatica

REPORT*RAPPORT*

Editing tree structures

L.G. Barfield

Computer Science/Department of Algorithmics and Architecture

CS-R9264 1992

Editing Tree Structures

Lon Barfield

CWI
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands
Email: Lon.Barfield@cwi.nl

Abstract

An interactive editor that operates on a projection of a tree structure can offer the user many possible commands. This document attempts to catalogue and classify, in a rigorous way, the commands that can be offered. The discussion starts with a simple tree and then moves on to more complex trees. It is assumed that the reader has some familiarity with the basics of structured editors.

The
V I E W S
System

1991 Mathematics Subject Classification: 68P05, 68U99.
1991 CR Categories: E.1, I.7.2.
Keywords and Phrases: tree editing, focus movements, structure editing, data structures.

1 Introduction

A structural editor operates on a model of some object (such as a text file). It is aware of the underlying structure of the object being edited and it offers the user a range of structure oriented tools to assist in the manipulation of the object.

There are several structural editors already in use, along with proposals for more comprehensive structural editors. Each addresses the problem of the command set in a rather ad-hoc fashion. This document approaches the area of the command set in a more rigorous way to see if this yields a better set of commands.

The terminology used in this document is summed up in figure 1. There are three different representations of an object. The first is the data of the object itself (the *structured object*), the second is the data represented as a *tree structure* and the third is the visual representation of the structure on the screen (the *presentation*). Converting the structured object to a tree structure is termed *modelling* and converting the tree structure into an on-screen representation is termed *presenting*. It is assumed that the reader is familiar with tree structures and some of the concepts behind structural editors such as the focus.

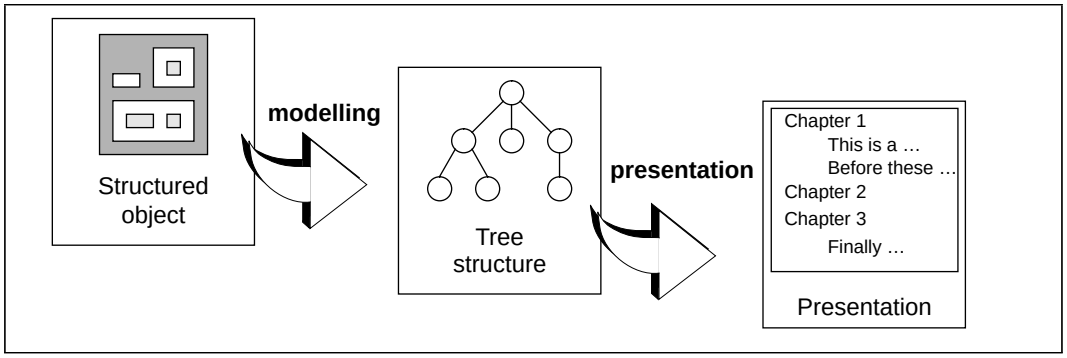

Figure 1 Tree structure terminology.

2 Classifying commands

Existing structural editors, usually text based, present a list of commands and their effect in terms of the text object being edited. This is highly application dependent. A more rigorous method is needed for the consideration of commands.

As yet I can think of only two guidelines applicable to designing commands, the first is that all possible configurations of the tree structure must be accessible using the command set and within an appreciably short time, thus. repeated use of some incremental command is prohibitive if it needs to be used a large number of times to accomplish one goal. The second is, that in order to ensure an easily grasped user model, it is better if the commands are grouped in pairs according to effect and counter effect, for example ‘up’ versus ‘down’, ‘move-left’ versus ‘move- right’, and so on.

2.1 Navigation and manipulation commands

In order to start thinking about the sort of commands that should be available it is helpful to first sub-divide the set of possible commands into more manageable groups. One method of sub-division is to split the commands up as below:

- **Navigational commands**
  These are commands that deal with altering the extent of the focus and moving the focus to different parts of the structure.
- **Manipulative commands**
  These are commands that deal with actually altering the tree structure. Adding, deleting and moving objects around.

However, all manipulative commands are combinations of ‘new’ and ‘delete’, possibly with some navigation of the focus in between. It would thus appear to be sufficient to consider just the ‘new’ and ‘delete’ commands.

2.2 Levels of commands

There appears to be another method of classifying commands that is orthogonal to the one above. This is to split up the commands according to the level of abstraction between the tree structure and its presentation on the screen. This gives rise to three classifications of commands:

- Structural commands
- Leaf relationship commands
- Presentation dependent commands

2.3 Structural commands

The most abstract class in this approach is made up of commands that are purely structural in nature. Their behaviour can be expressed explicitly in terms of the trees structure. As such they are extremely general and can be applied to any tree structured object regardless of the object that is being modelled. (They will be discussed in more depth later on.)

2.4 Relationship dependent commands

In a tree structure there are the structural relationships between nodes (parent, child, sibling and so on.). There are also extra relationships associated with the elements that are dependent upon the object that is being modelled. Usually these relationships are dependent in some way upon how the structure is ordered in one or more dimensions.

- **Zero dimensional**
  A tree-structured file system has no dimensional relationships between the nodes. The tree structure is the only factor. The alphabetic ordering is purely part of the presentation process.
- **One dimensional**
  Text processing deals with a one-dimensional structure; a one-dimensional stream of characters, you can navigate through it by going forwards or backwards. The move cursor up and down commands are again only connected with the way in which the structure is presented.
- **Two dimensional**
  Two-dimensional relationships are very common since this is the natural method of communication using printed documents. Text formatting systems (incorporating text ordering *and* two-dimensional layout) and simple graphics systems concern themselves with two-dimensional relationships.
- **Three dimensional**
  As two-dimensional graphics above but ordered in three dimensions.

- **Higher dimensions**
  Tree structures can be used to represent more abstract models with n-dimensional relationships and possibly dynamic structures as well.

Navigation consists of either extending or moving the focus. The two key factors are therefore direction of extension/movement and any boundaries on the extension/ movement. Both of these must be expressed in the dimensionality of the structured object. Movement in a direction can be incremented upon either position or element as there are two classes of structure; dense and sparse:

- **Dense**
  In a dense structure most of the possible positions in the structural space are occupied by elements. Moving through the structure is best done by incrementing upon position, (‘Move to the adjacent position in *this* direction’). Examples of dense structures are text (one dimensional) and pixel-based patterning packages (two dimensional). In both cases there are very few empty positions.

- **Sparse**
  In a sparse structure only a very small fraction of the possible positions in the structural space are occupied. For example two-dimensional and three-dimensional line drawings or textual modelling of structures where spaces are important. Moving through a two-dimensional drawing by incrementing on position would be very time consuming (move to the adjacent co-ordinate over and over again). Increment by element is used in these situations to navigate around the structure (‘Move to the next element in this direction’).

Manipulation is composed essentially of ‘delete’ and ‘new’ as outlined earlier. Deleting what is in the focus needs no structural information or leaf relationship information. However, a new node cannot be specified in terms of leaf relationships alone, so when creating a new node two parameters have to be specified; its position in the tree structure and its position in the structured object relative to other elements. (This dual parameter problem is removed to some extent by tree structures where the position of the element in the structured object is implied by its position in the tree structure and vice-versa.)

2.5 Presentation commands

These commands deal purely with the way the tree structure is presented on the screen, commands such as ‘point and select’, ‘up/down display line’. As with leaf relationship commands projection commands only seem to play a role in navigation.

- **Up/down/left/right**
  These commands move the focus about on the two-dimensional surface of the screen, whether these commands are truly presentation dependent or not depends upon how the structure is presented onto the screen, In a text based system the up/down display line is a true projection dependent command, left/right however is a leaf-relationship command as it is just moving the focus forwards and backwards through the one-dimensional structure of the text.

- **Pointer interaction**
  Interacting using a pointer (mouse) is, in abstract terms, the same as the above, the only difference is in the hardware.

- **Selection**

- Selecting (or positioning the focus) using the above is a very fast and useful method of specifying nodes. However, in a tree structure where only the leaf nodes are elements difficulties may arise when the user wants to point at nodes higher up the tree. For example when pointing at text the user may want to focus on the character the pointer is over, or the word, or the sentence and so on. The

granularity of this operation must be decided upon and the user can then modify the selection using up structure or down structure.

2.6 Conclusions

Leaf-relationship commands and presentation commands seem to be so entrenched in application dependency that it is difficult to extract any useful generalisations from them and the investigation will continue by considering only true structural commands.

3 Tree structure complexity

The underlying structure is assumed to be a tree structure. This is the case in the majority of applications and it is difficult to envisage other structures that would be as useful. The investigation of the structural commands will start by considering a very simple tree structure and the arguments developed will then be applied to successively more complex tree structures.

The initial structure will be a tree structure with each node having the potential for n - children, the children are unordered, only the leaf nodes are elements of the structure (other nodes represent the structure within which these elements reside) and the levels within the tree are homogeneous, i.e. everything on the same level is the same type of node.

Once this simplest case has been dealt with the tree structure will be made successively more complex and the same issues of command design will be re-examined.

- The first complexity will be to introduce order to the tree.
- Next all nodes (not just leaf nodes) will be elements.
- Thirdly the levels within the tree will be non-homogeneous (i.e. nodes on the same level can be of different types).
- Finally several minor complexities will be introduced such as cross-links and references. A cross-link is a reference from one part of the tree structure to another. They can be such things as symbolic links in a file system or references and indexes in a text document. A floating object is a node that is relevant to the structure and may be referenced, but might not be confined to a fixed place in the projection of the structure, (in a textual structure floating objects are such things as diagrams and tables included in the text).

In the following command descriptions, diagrams will be used to clarify the text. The diagrams will be abstract and show a before and after representation of the tree structure. Commands will be expressed as simply as possible. Composite commands are possible (such as ‘Create a new node and move the focus onto it’) but as there are so many possible composite commands only a few will be discussed.

Permissible focus structures within the tree will also be discussed and prerequisite focus structures for each commands will be mentioned as necessary

4 Structural commands - simple tree

In a simple tree structure every focus must include the leaf nodes since they are the only nodes that are elements in the structure. Thus the focus either consists of several adjacent levels including the lowest level or, a special case of this, just the leaf node level, see figure 2.

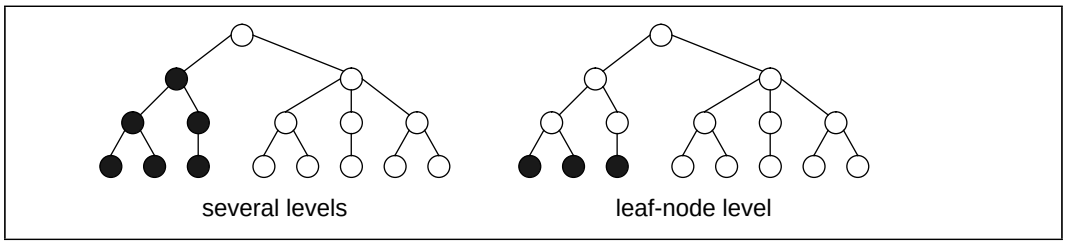

Figure 2 Foci in a simple tree.

The top level of the focus must be either a single node, all the nodes in a sub-group or all the nodes in a level, see figure 3.

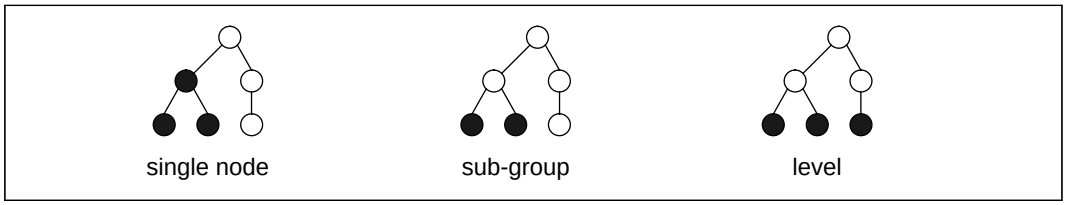

Figure 3 Top level configurations of the focus.

In a simple tree the only connection between two sibling nodes is the parent node, and so a situation such as that shown in figure 4 (the left-hand tree) is in fact two separate foci, one on node A and the other on node B. This can easily be illustrated since it is allowable to redraw them to show them as two separate foci (right-hand tree). The apparent order of the sibling nodes is just due to the way that the tree is drawn. This ambiguity does not arise with an all-sub-group top-level focus or an all-level top-level focus.

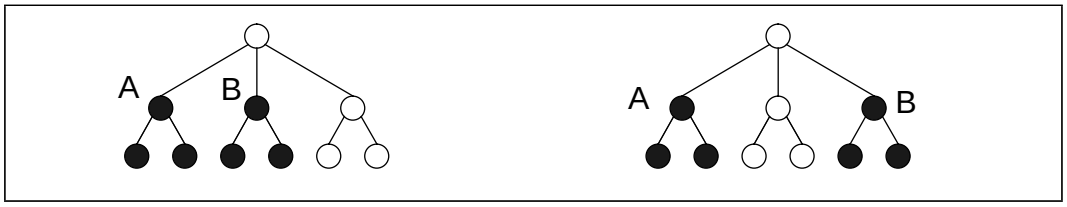

Figure 4 Two foci looking like one and then redrawn to show them as two.

There appears to be four definite structural commands for navigating through the structure and altering the scope of the focus (see figure 5):

- **Down structure**
  This moves the top level of the focus down to the next level.

- **Up structure**
  'Up structure' moves the top level of the focus up to the next level. If the top level is a single-node top-level then all other sibling nodes and their associated sub-trees are included in the focus.
- **Widen to limits of sub-group**
  This accepts a focus with a single node top level and extends the focus to include all its siblings (and their sub-trees). It can be argued that this is a composite of the two previous commands since 'up structure' followed by 'down structure' will have the same results.
- **widen to limits of level**
  This is similar to above but the focus is extended to include all nodes in the level. Again it could conceivably be a composite command repeated use of 'up structure' until all the tree is in the focus, followed by the same number of 'down structure' commands would have the same effect.

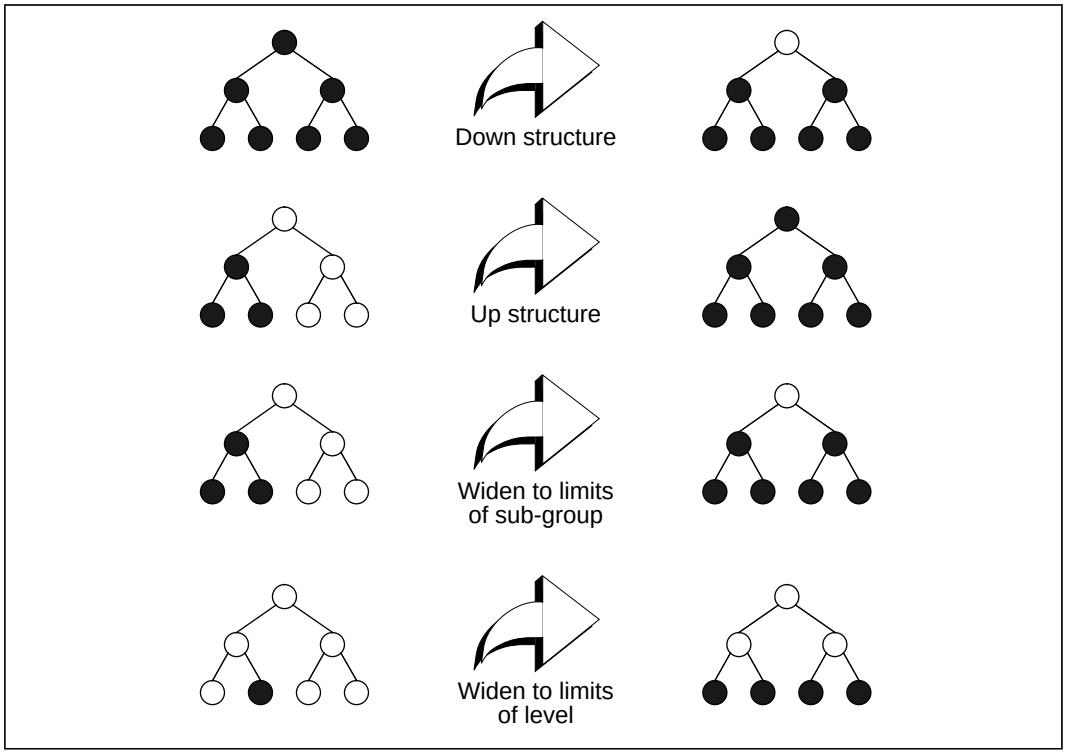

Figure 5 The four structural commands.

Deleting a node only involves specification of the node to be deleted, and introducing a new node into an unordered tree structure involves only the specification of the parent or the level of the new node, thus:

- **Delete**
  This deletes the node(s) that are currently in the focus, see figure 6.
- **New**
  The position of a new node in the structure can be specified in two ways, either a single-node top-level focus can be taken to mean create the new node as a child of *this* (the top) node. Or an all sub-group top-level focus can be taken to mean

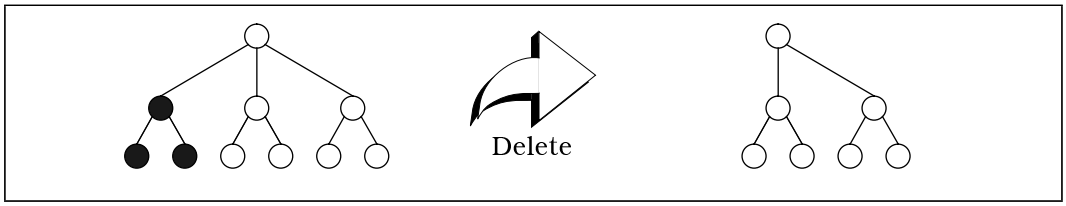

Figure 6 The ‘delete’ command.

create the new node within this sub-group (see figure 7). An all-level top-level focus is not able to simply and un-ambiguously specify where the new node is to be appended. Now that the position has been identified the actual creation of the new

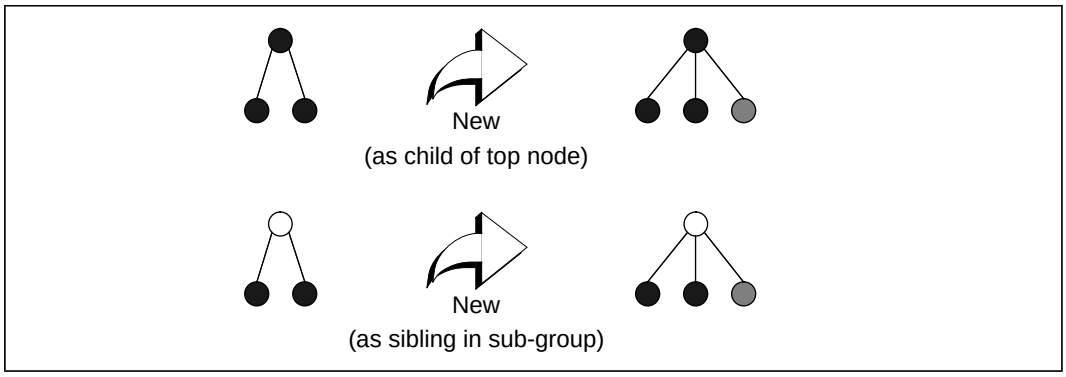

Figure 7 The ‘new’ command

node takes place. Since leaf nodes are the only elements the new node must have extra structure included down to the leaf node level to allow the user to specify the element that is to be in the leaf node. The node could be created as in figure 8. Further leaf nodes can be created using ‘new’ to create siblings to the existing leaf node.

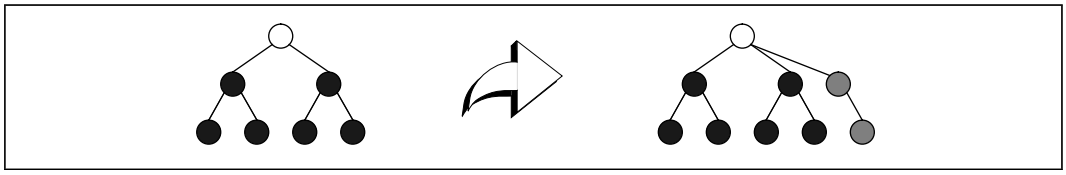

Figure 8 When creating the new node, extra structure is added.

5 Structural commands - ordered tree

Ordered Tree Structure The above section deals with tree structures in their simplest form. The next step is to apply this same methodology of command classification to ordered tree structures. In an ordered tree structure all the children of a node are ordered in one dimension. In the abstract diagrams the order is taken to be left to right. This extra detail

leads to a large number of new commands. Because the children of a node can now be addressed individually using just the structure, new classes of foci emerge. Previously foci could be *single node* top level, *all sub-group* top-level or *all-level* top-level (see figure 9).

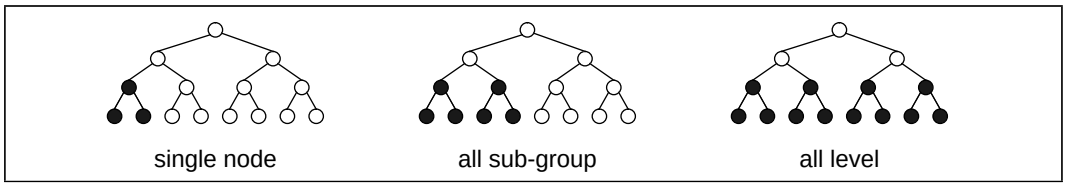

Figure 9 Classes of foci on top level in non-ordered trees.

Now the top level can occupy part of a sub-group yielding *partial sub-group* top-level or part of a level (where the part cannot be defined by just sub-groups) yielding *partial-level* top-level (see figure 10). As one might expect the command set is more complex:

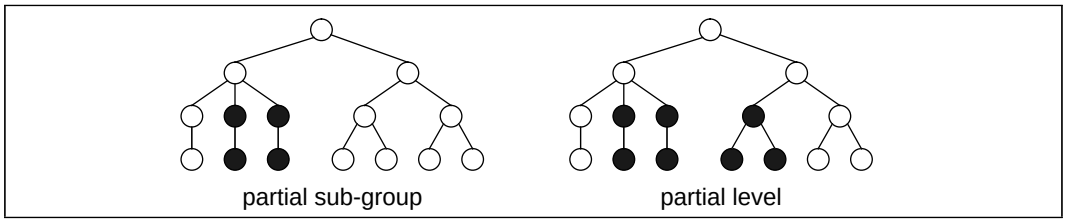

Figure 10 Two extra classes of foci on top level in ordered trees

Up structure

This moves the top level of the focus up to the next level. The new top level should include the parents of all the nodes in the previous top level, and the new focus should include all the sibling nodes (and sub-trees) of the nodes in the previous top level. This is a more general statement of the conditions in the previous up structure command. As an illustration consider figure 11.

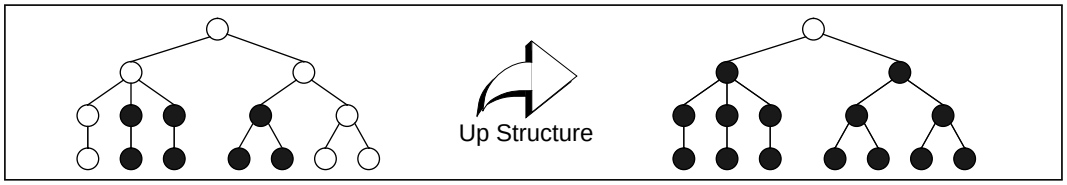

Figure 11 Up structure in an ordered tree.

Down structure

This behaves as in the previous section.

Down structure to n-th child

This moves the top level of the focus down and narrows it so that it is confined to one of the child nodes. See figure 12 for an example. Its prerequisite is a single node top level focus. It would seem a complex command involving the user having to be aware of the

numerical position of each node amongst its siblings. The command would be more useful if it were expressed as two special cases; ‘down structure to first’ and ‘down structure to last’. However, if this were the case the access to all configurations rule identified in section would mean that other commands would have to exist to get from the first or last node in a sub-group to other nodes in the sub-group.

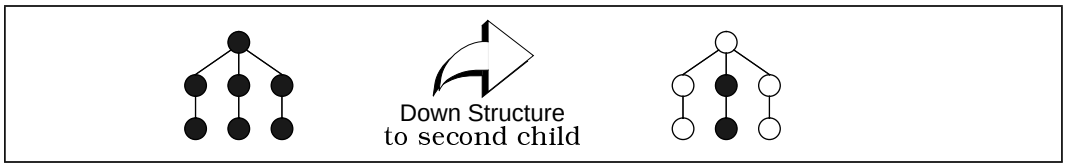

Figure 12 Down structure to n-th child.

Move focus within this level

In a one dimensional ordering with limits the focus can be moved in two different ways; either incrementally through the nodes or in one jump to the node at the limit. There are two sets of limits (limits of sub-group and limits of level, see figure 13) and two directions of movement. Overall this gives eight possible commands (2x2x2). It is assumed the familiarity with the diagrams used so far and the information above will suffice for the behaviour of these commands.

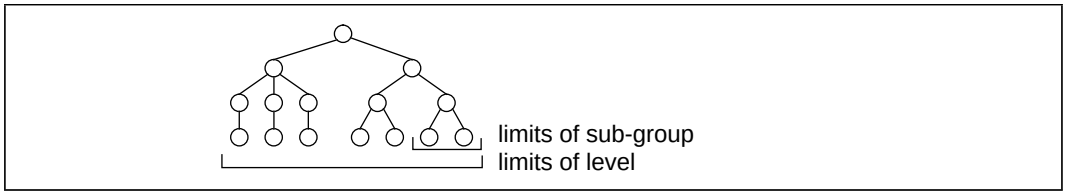

Figure 13 The limits in an ordered tree.

The first four commands take a single node top level focus as a prerequisite:

- Previous node - sub-group
- Next node - sub-group
- Previous node - level
- Next node - level

While the last four will accept any of the five focus configurations:

- First node - sub-group
- Last node - sub-group
- First node - level
- Last node - level

The question arises of how the incremental movement commands should behave upon encountering the limit to which they are confined. Either the focus could remain on the node at the limit and further incremental move commands in that direction would have no

effect (except maybe an error beep). Alternatively the focus could cycle through the group or level confined by the limits, i.e when it reaches the node at the limit the incremental move in that direction would take it to begin again at the other limit of the group or level.

Extend focus within this level

The case with extend is similar to move, and again there are eight possible commands. All eight commands accept any of the five focus configurations:

- Extend forwards - sub-group
- Extend backwards - sub-group
- Extend forwards - level
- Extend backwards - level
- Extend forwards to end - sub-group
- Extend backwards to beginning - sub-group
- Extend forwards to end - level
- Extend backwards to beginning - level

The behaviour of extend at the limits of the group/level is trivial - the extension stops there and further extend commands in that direction have no effect.

Contract focus

Contract has only four possibilities, it can either contract the front or the back of the focus in an incremental fashion. Thus:

- **Contract front**
  (see figure 14)
- **Contract back**
  As contract front but operates on the back of the focus.

Or it can contract to a specific node within the focus. Just as with down structure to child the two special cases are probably the most useful:

- **Contract to first**
  (see figure 14)
- **Contract to last**
  As above but contracts to the last node in the top level of the focus.

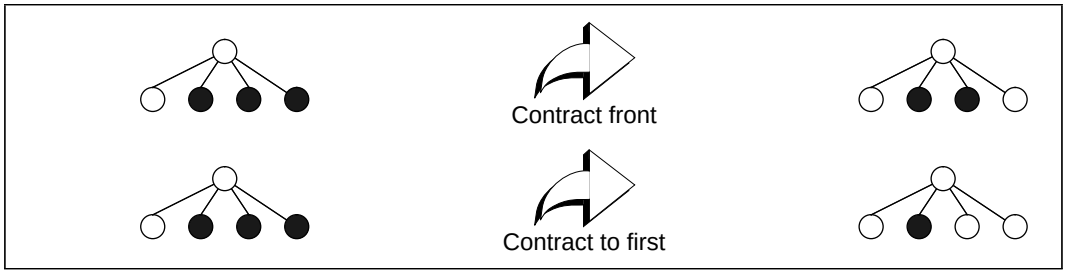

Figure 14 Contracting the focus.

As well as the navigation commands there are of course the manipulation commands:

- **Delete**
  As before this simply deletes everything in the focus.
- **New**
  In the previous contexts new child of this node or new sibling in this level was sufficient. Now two things need to be specified; the parent node/level of the new node and the position amongst its siblings that it must take. Since the first is implicit in the second the position of the new node can be specified by focusing on one of the sibling nodes and new could then insert the new node in front of or behind the node in the focus. To take all the other focus configurations into account two forms of the new command are suggested:
- **new - front**
  This inserts the new node just after the front most node in the focus. As shown in figure 15.
- **new - back**
  Similarly this inserts the new node at the back of the focus. As before extra structure is inserted down to the leaf node level.

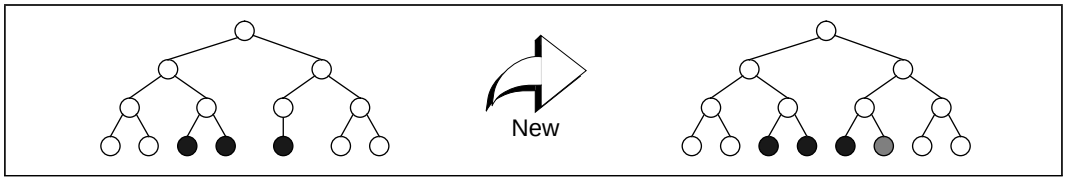

Figure 15 New in front of focus.

6 Structural commands - nodes as elements

The next complexity is that not just leaf nodes are visible elements of the structure. Nodes further up the tree, as well as being groupings of the leaf nodes, have tangible features of their own. Previously there were five focus configurations classified according to the top level of the focus. Now the bottom part can also have different properties. This gives us another two focus classifications that are orthogonal to the other five:

- Multiple level foci
- Single level foci

All previous navigation commands operated on the top level of the focus now extra commands are need to operate on the bottom of the focus (see figure 16).

- Bottom level down
- Bottom level up

Extra commands are also needed to deal with the special case where the focus is a single level:

- Up structure - single level
- Down structure - single level
- Down structure to first - single level
- Down structure to last - single level

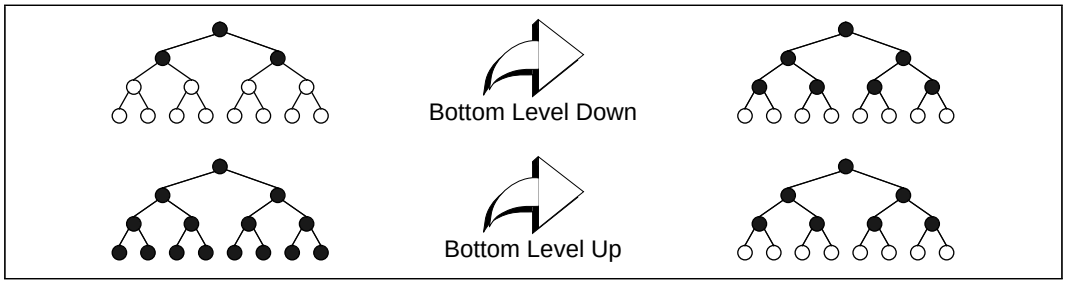

Figure 16 Commands on bottom of focus.

Two more possibly useful commands go from multiple level foci to single level foci (see figure 17):

- Contract focus to top level
- Move focus to leaf nodes

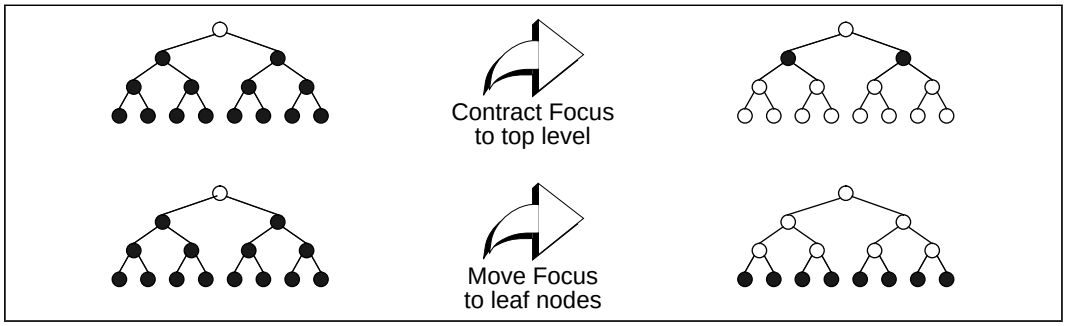

Figure 17 From multiple level foci to single level foci.

Now the manipulation commands:

- **Delete**
  Deleting a node and associated sub-trees is as in previous sections, the new aspect of delete is deleting high level nodes only. Consider the situation in figure 18. A node is in the focus and its sub-tree is not. When the delete command is issued the node will be deleted, but what will become of the sub-tree? It cannot be appended to the parent of the deleted node since this would violate the homogeneity of the tree. One solution is to append the sub-tree (maintaining its order) to one of the nodes adjacent to the one deleted.

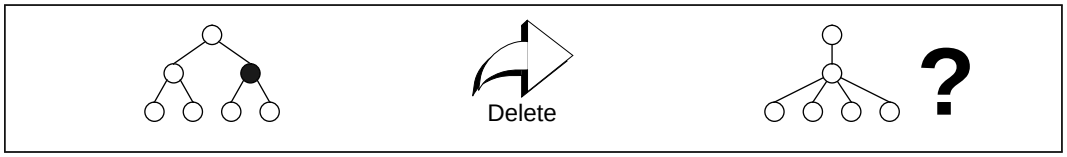

Figure 18 Deleting a non-leaf node.

- **New**
  The new command actually becomes somewhat simpler! It behaves as in the previous section but the complication of adding extra structure down to the leaf nodes is no longer needed. See figure 19.

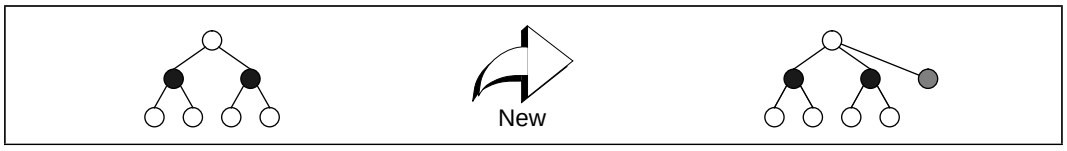

Figure 19 Creating a new non-leaf node.

7 Non-homogeneous levels

Up to now all the tree structures have been homogeneous, i.e. everything on the same level in the structure is of the same type. The best example of this is a structured text document. The characters can only exist in the bottom level, words in the next level and so on. In non-homogeneous tree structures elements of the same type may be on different levels and sub-trees of a node may be of very different structures (e.g. text and graphics in a document).

This causes great complications with navigational commands that involve traversing the structure (e.g. move focus, extend etc.). How should they behave when they encounter a boundary between two differently structured parts of the tree?

There are many approaches to this problem, one is for the extend/move focus commands to behave as before when operating within the limits of a sub-group. Consider a mixed text and graphics document, moving the focus incrementally at a high level through paragraphs and figures would move the top node (assuming it was a single node top level focus) from paragraph to paragraph to figure to paragraph and so on. Similarly with graphic elements and text elements in a figure (see figure 20, first tree).

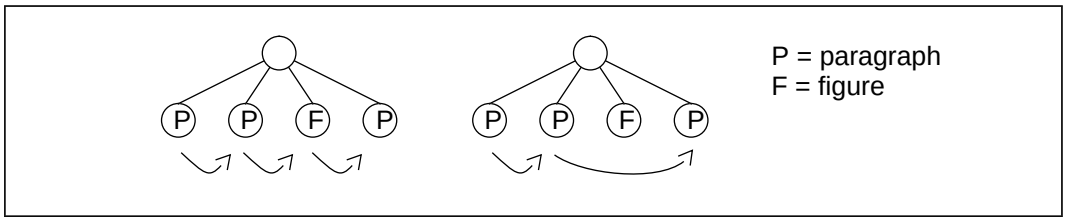

Figure 20 Two ways of moving within a sub-group of mixed structure.

Moving across a level may be confusing at boundaries in a mixed structure. Consider moving through sentences and then through elements of a picture. One could confine movement within a level to only move through nodes of the same type as the node that was started on, thus skipping parts of the structure that were different. (See figure 20, second tree.) The user can then deal exclusively with the text. Moving to the figure could be accomplished by moving the focus until its top node was in the same sub-group as the figure (i.e. to the paragraph level). The focus could then be moved onto the figure and moved back down the structure. (Another way would be to reposition the focus with a pointing device.)

The above deals with mixed structures, what about non-homogeneous structures? Consider figure 21 where the letters indicate the type of the node. Thus C_3 is the same type as C_1 and C_2, but it is on the same level as the different type B.

Figure 21 A non-homogeneous tree.

The boundary problem could be dealt with as above; going up structure to a common node and then coming down the structure, (i.e. jumping over the boundary). An alternative is to allow the traversing but carry it out either by level or by type. Thus if the focus were on C_1 a move focus right (by level) command would move it to B. While a move focus right (by type) would move it to C_2. This whole area is in need of a lot more discussion and may benefit from being treated as an issue in its own right. Manipulation commands have problems associated with changing the structure so that it is consistent with the grammar that describes it. In other words keeping the structure legal.

- **Delete**
  Deletes everything in the focus. If there are sub-trees below the focus that need re-appending after delete they can either be appended directly to the parent of the deleted node, see figure 22, or as before they can be appended to an adjacent sibling node. However inconsistencies may arise in situations where the adjacent node is of a different type and may not be able to accept the appended nodes.

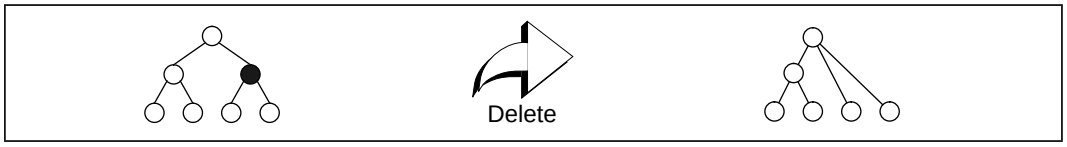

Figure 22 Re-appending a sub-tree to its grand-parent.

- **New**
  Previously the type of a new node has been implicit in the level that the new node is being created on. Now the new commands needs to know what the type of the new node is as well as where in the tree it should go.

Successful operation of the command therefore depends on the type of the node being created and whether the grammar allows that node to exist where the user is trying to place it.

8 Structural commands - other complexities

Two complexities that have come to light so far are cross-links and floating objects. A cross-link is a structural link that violates the usual tree structure in that it can cross over to other sub-trees and link to nodes in other levels. Consider the example in Figure 23. The sentence S contains some words and a reference to a figure that is in the same level as the paragraphs of the section.

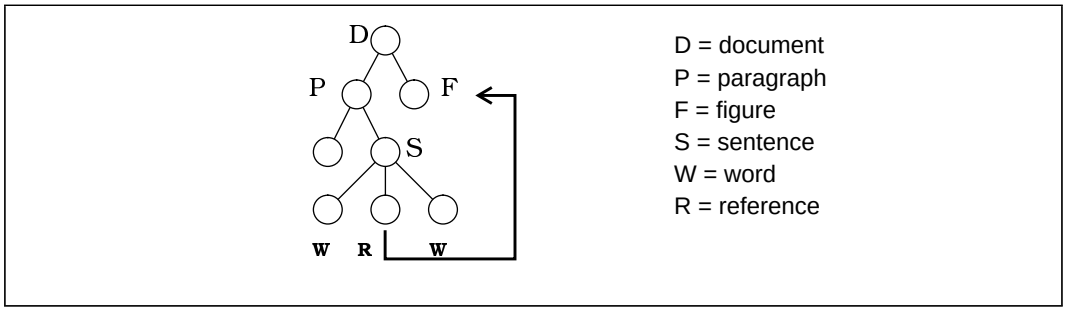

Figure 23 Cross-link in a document.

Commands applicable here would seem to be

- **Create cross-link**
  This would either need two separate foci or the procedure could be; place focus on reference node, specify start cross-link, move focus to figure node, specify end cross-link.
- **Move focus via cross-link**
  Assuming that the focus in some way implies the reference node or the figure node (e.g. a single node top level focus with the top node on the node in question), this command would move the focus to the other end of the cross-link.

The figure could be a floating object or floating node in that when the structure is projected it may not be in the position amongst its siblings that it was in the tree structure, see figure 24. The move/extend focus commands could then have behaviour that would be confusing to the user. One solution could be to have move/extend commands ignore floating nodes and have some way that the user can expressly move the focus to them, this may already be taken care of with the follow cross link command although this may be a very round about way of focusing on something.

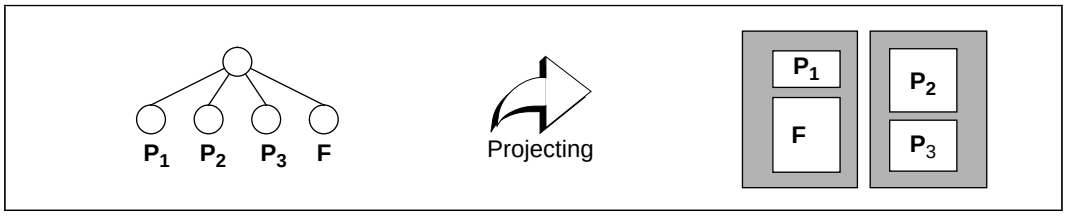

Figure 24 Floating nodes in a document.

It is assumed that there are further complexities that are not covered here. However, as they are issues that are separate to the more general commands sets they can be dealt with as they arise.

9 Conclusions

This document attempts to catalogue the commands desirable for operating on a tree structure. From this set commands could be selected for a variety of purposes; an actual operating command set could be constructed, or single commands could be considered in various specific contexts to discuss their behaviour.

Due to the general nature of the document the coverage is deep but not broad, navigation and manipulation commands have been thoroughly dealt with but there may exist other, more specialised commands, that play an equally important role, for example ‘visit’.

Finally the use of the before and after diagrams of tree structures and the descriptions of focus configurations has led me to wonder whether it would be possible to design a rigorous and structured language with which to specify the effect of commands upon the focus and the tree structures. The definition of the behaviour of the commands could then be represented by a document which would itself be an editable part of the system.