

A Blast from the Past: Using P-EDIT for Multidimensional Editing

Vincent Kruskal

IBM Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598 USA
+1 914 784 7645
kruskal@watson.ibm.com

1 INTRODUCTION

In the 1980s, a few colleagues and I developed a text editor, P-EDIT, designed to permit viewing and modifying text files where the contents were organized into multidimensional spaces. Except for the multidimensional feature (and an UNDO feature), it is a typical VM/370 command line text editor for its day. The command set included LOCATE, CHANGE, NEXT, UP and INSERT. The display is used to show the lines around the "current line". It is still used today by a very few, including myself.

It should be of interest to the Multidimensional Separation of Concerns community for two reasons:

1. P-EDIT is an effort a decade or two earlier which showed essentially the same concepts expressed in quite a different way. This might lead people to see new ways to express hyperspace concepts.
2. P-EDIT acts on a smaller granularity than many MDSOC proposals. P-EDIT acts at the level of lines in a text file, not that of classes and their methods and fields.

2 A BRIEF INTRODUCTION OF P-EDIT

A file in P-EDIT is a list of pairs of text and a Boolean expression. The text is usually a line of source code, although P-EDIT is also used to maintain documents written in a markup language. The Boolean expression is made of relations of the form, DIMENSION=CONCERN and DIMENSION^=CONCERN. These relations are connected together in the usual way using parentheses and AND (&), OR (|) and NOT (^). The Boolean expressions are largely created and manipulated automatically and hidden from the user.

One view of a P-EDIT file is that it defines a set of "instantiations". Each instantiation is a text file. It can be selected with an instantiating Boolean expression. We also talk of the instantiations of a block within a P-EDIT file. In general, this will be a smaller set than the file's instantiations because of dimensions and concerns having no bearing on that block.

When editing a P-EDIT file, the user normally chooses just a hyperslice of it to operate on. The hyperslice is controlled by a Boolean expression called the "effective mask". In order to give the user practical control over which hyperslice he is editing, there are named masks and a mask stack. Masks can be assigned a Boolean expression, usually a single relation, and can be turned on and off. Boolean expressions can be pushed onto the stack and popped off. The effective mask is the AND of all those masks which are turned on.

A line is in the current hyperslice if its Boolean expression is consistent with the effective mask. If it is implied by the effective mask, it is "fixed". If merely consistent, it is "unfixed". The other lines are "excluded" and no editing action will have any effect on them nor will the user be able to see them. If the user searches for text, he will be able to find it only in fixed and unfixed lines.

The most common way for the user to set masks is "through the text". He can issue a command to set a mask so that an unfixed line becomes fixed (FIX). Conversely, he can set the mask so that it becomes excluded (EXCLUDE). He can also start a loop where P-EDIT sets a mask to cycle through each instantiation of a block of lines (SHOW, STEP, STEP, STEP, ..., UNSHOW). Of course, the body of the loop is the user actions. In practice, this provides a very effective multi-dimensional scrolling ability.

Whenever the user modifies a line of text, P-EDIT first duplicates the line into its excluded and non-excluded parts. So if the effective mask is SYSTEM=WINDOWS and a line to be modified has the Boolean expression DEBUG=ON, the excluded copy would have a Boolean expression of DEBUG=ON & SYSTEM^=WINDOWS and the copy to be modified would have the Boolean expression of DEBUG=ON & SYSTEM=WINDOWS. (If the excluded line's Boolean would be FALSE, it is not created.

Example file as P-EDIT sees it

```
public String appendDirectoryToFileName(      TRUE
        String directory,                    TRUE
        String fileName) {                  TRUE
    if (Debug.in("appendDirectoryToFileName")) { DEBUG=ON
        System.out.println(                 DEBUG=ON
            "appendDirectoryToFileName returning: " DEBUG=ON
            + directory                      DEBUG=ON
            + '/'                            DEBUG=ON
            + '\ '                           DEBUG=ON&SYSTEM^=WINDOWS
            + fileName);                    DEBUG=ON&SYSTEM=WINDOWS
        }                                  DEBUG=ON
    return                                  DEBUG=ON
        directory                            TRUE
        + '/'                                TRUE
        + '\ '                               TRUE
        + fileName;                          TRUE
    }                                        TRUE
}
```

There are two ways to insert a line, INSERT and APPEND. INSERT inserts it into the total hyperslice, so it would have a Boolean of DEBUG=ON. APPEND inserts it into only the subhyperslice which the previous line is in, so if that line has a Boolean of SYSTEM=WINDOWS, the inserted line would have a Boolean of DEBUG=ON & SYSTEM=WINDOWS.

So what does the user see on the screen? We could have shown him all non-excluded lines along with their Boolean expression simplified relative to the effective mask (that is, simplified with the NOT of the mask taken as a "don't care"). That would look somewhat like a C program with a lot of preprocessor #IFs. Quite confusing. In fact it was just such confusion that was a chief motivator for the development of P-EDIT. But even with this "flat" display, P-EDIT would still have had the advantage of letting the user make simplifying assumptions by setting masks so that the confusion would be under better control than editing such a C program.

But we chose to always display a single instantiation, the "view". Any lines in the view which are unfixed, are highlighted. If there are a sequence of unfixed lines not in the view bracketed by two fixed lines, so that there is nothing to highlight, the fixed line above them is highlighted if they are before the current lines or the one after them if after the current line. If the current line is not in the view, it is shown as a highlighted blank line.

The user can control the view in similar ways as he can control the effective mask. He can ask that a displayed unfixed line not be in the view (HIDE). He can ask that a non-displayed unfixed line be in the view (UNHIDE). And he can simply specify a Boolean expression (VIEW) to use. That Boolean need not specify one instantiation; if needed P-EDIT would refine it using a history of prior views in an attempt to decrease abrupt changes. Finally the user can loop through all possible views of a block of lines (VIEWSHOW, STEP, STEP, ..., UNSHOW).

Example file as P-EDIT sees it with mask SYSTEM=WINDOWS and view DEBUG=ON

```
public String appendDirectoryToFileName(
        String directory,
        String fileName) {
    if (Debug.in("appendDirectoryToFileName")) {
        System.out.println(
            "appendDirectoryToFileName returning: "
            + directory
            + '/'
            + '\ '
            + fileName);
    }
}
```

Highlighted
Highlighted
Highlighted
Highlighted
Highlighted

```

+
+ '\ '
+
+ fileName);
}
return
directory
+
+ '\ '
+
+ fileName;
}

```

```

Highlighted
Highlighted
Highlighted
Highlighted
Highlighted

```

Scripts:

P-EDIT supports a VM/370 general interface to scripting languages. This can be used to write macros and quite a large collection are considered part of the system. But most importantly, a script, selected by the "type" part of a file's name, can be used to set masks as appropriate for that set of files. Masks and other multidimensional control information are normally private to each file being edited. But such file initialization scripts can make it such that all files edited in the same group share the same multidimensional control information.

The Unnamed Concern:

If a dimension, X, has concerns 1 and 2, then by Boolean logic, there is a concern with no name which can be seen by setting a mask to $X^{\wedge}1 \ \& \ X^{\wedge}2$. This unavoidable consequence of this Boolean scheme has both an advantage and disadvantage. The disadvantage is that one of the concerns can be quite hard to get to. Users, of course, have the option of setting a mask to $X=1 \ | \ X=2$ and forgetting the unnamed concern exists. This mask setting could be automated in the file initialization script.

Declaring New Dimensions and Concerns:

There are no declarations in P-EDIT. Simply by setting a mask to a relation involving a new dimension or concern (and modifying the file), it will exist. Say there is a line in all instantiations of the file (Boolean expression of TRUE). He sets a mask to a new dimension (MASK A NEW=YES) and modifies the line. Now there will be an excluded line whose Boolean is $NEW^{\wedge}=YES$ and a modified line with a Boolean of NEW=YES. Similarly, he might set a mask to mention a new dimension (MASK A SYSTEM=NEW). Before he makes any changes, what will the hyperslice he is editing look like – the unnamed version. Of course, once he makes a change, the new SYSTEM=NEW hyperslice will go its own way, leaving the unnamed version behind. So, in a system with no declarations, an advantage of having an unnamed version is having a default version.

But what if he wants, instead, the new concern to initially look like some other concern which he would like to specify? For that, the MAKE command is provided. MAKE is an editing command which explicitly modifies

the Boolean expressions associated with a block of lines. Syntactically it is a takeoff on the text modifying command, CHANGE/original/new/block, where "block" specifies how many lines are to be changed. To add a new concern to SYSTEM which originally looks like SYSTEM=WINDOWS, the user goes to the top of the file and issues MAKE SYSTEM=WINDOWS SYSTEM=WINDOWS | SYSTEM=NEW *. The MAKE command will go through all the Boolean expression in the file looking for occurrences of SYSTEM=WINDOWS or $SYSTEM^{\wedge}=WINDOWS$ and change each to (SYSTEM=WINDOWS | SYSTEM=NEW) or (SYSTEM^=WINDOWS & SYSTEM^=NEW), respectively.

Numeric Concerns:

There are two common dimensions where it is convenient to have the concerns taken numeric values. They are TIME and RELEASE. It is common for the file initialization script set a mask so that changes are time-stamped and old versions can be viewed and, even, resurrected. The command which it would use is MASK TIME TIME>=CURRENT(TIME), illustrating that even Boolean expressions can call out to a script, CURRENT. If issued as I write this, it would set the TIME mask to TIME>=2000.02.16.14.53.22. Two features were added to support numeric concerns. One is to complete the relational operators with <, <=, > and >=. The other is to interpret concern names as base 36, dewy-decimal numbers. Thus alphanumeric concern names, time and many common version naming techniques are supported in one typeless (or single type) language. Adding the other relations is important, not only as a shortcut for Booleans with long sequences of ORs (TIME=2000.02.15.02.44 | TIME=2000.02.16.14.53.22 | ...), but to permit new TIME concerns to be added without issuing a MAKE command each time the file is edited.

The MAKE command does have an interesting use with regard to numeric concerns. It can be used to resurrect old versions. For example, MAKE TIME=1999.1.2 TIME=1999.1.2 | TIME>=CURRENT(TIME) over some block of lines, will bring back that block as it was January 2, 1999. Of course, for this to have any effect, the TIME

mask must be turned off first.

3 CONCLUSION

P-EDIT can be used to express any separation of concerns within a body of code. But where it shines is where more than one dimension of concerns bear on the same lines of code.