File No. UIUCDCS-F-85-934

Editing Network-Structured Knowledge Bases in the ADVISE System

Thomas D. Channic

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

February 15, 1985

ISG 85-4

ABSTRACT

Until now, knowledge bases in the ADVISE meta-expert system have had to be edited by text editing or by changing the program that creates the knowledge base. This paper discusses a network editor for ADVISE knowledge bases, which can edit a knowledge base directly. The network editor has been implemented on a Sun Microsystems Workstation, and is screen-oriented and menu driven. The latter part of this report serves as a user's guide for the editor.

This work was supported in part by the National Science Foundation under grant DCR 84-06801 and by the Office of Naval Research under grant N00014-82-K-0186.

Editing Network-Structured Knowledge Bases in the ADVISE System

1. INTRODUCTION

The ADVISE expert system (Michalski 84) provides a powerful framework for building expert systems. ADVISE has been used to build two very different expert systems - namely, BABY (Rodewald 84), a system used for patient monitoring in a newborn intensive care unit, and PLANT (Reinke 83), a system for diagnosis of soybean diseases. Until recently, however, ADVISE lacked a facility for directly manipulating and editing a knowledge base for expert systems. If a knowledge base needed to be changed, either a text file (which is frequently not an accurate representation of an ADVISE knowledge base) had to be edited or the program that created the knowledge base must also include code to alter the knowledge base. Thus the only way to alter the PLANT knowledge base was either to text edit backup knowledge base files or run a rule parser on new input rules. QUIN, a program for editing knowledge bases represented as relational tables (Spackman 1983), is available but has minimal benefit for the rules and network representations of currently implemented systems.

This report discusses the network editor, which provides ADVISE with the capability for interactive manipulation of knowledge bases in the ADVISE representation. A brief description of the ADVISE knowledge representation is provided below followed by a user's guide for the network editor.

2. REPRESENTING KNOWLEDGE IN ADVISE

The basic structure for representing knowledge in ADVISE is a *tuple*. A tuple is similar to a list in LISP, and differs mostly in that it is implemented in Pascal. Nodes are like LISP atoms, and a tuple is just a list of nodes. The second node of the the tuple has special

meaning as a relation or arc between the head node and subsequent nodes in a tuple. A typical tuple looks like the one below.

```
(headnode arc subnode1 subnode2 subnode3 ...)
```

Of course, the same head node can have many arcs (relations) under it. These can be represented simply by additional tuples as follows.

```
(headnode arc1 subnode11 subnode12 ...)
(headnode arc2 subnode21)
(headnode arc3 subnode31 subnode32 subnode33 ...)
```

For efficiency reasons the above tuples would be stored as below.

```
(headnode (
      (arc1 subnode11 subnode12 ...)
      (arc2 subnode21)
      (arc3 subnode31 subnode32 subnode33 ...)
))
```

In the actual implementation of this representation, nodes are memory addresses. Nodes also have printnames associated with them as well as being associated with the tuples in which they appear as head node. The ADVISE tuple manager (see the ADVISE Technical Document) handles all the manipulations of the knowledge base on the tuple level. The network editor simply makes the appropriate calls to the tuple manager based on its interaction with a user.

The tuple representation represents an important generalization over the basic concept of semantic networks (as described, for example, in Winston 84). Thinking about tuples in light of these networks, each tuple with the same head node can be considered a slot, each slot has

a name (arc) and a value. Slot/value combinations are also known in ADVISE as *attributes*. The generalization over other representations is that slots or attributes can have many values associated with them. Thus, similar slots can be combined into a single slot

(house ((has-room living-room dining-room bedroom kitchen)))

or a single slot may have several values associated with it, for example, both a qualitative and quantitative value.

(block-1 ((orientation vertical 89.5)))

The ADVISE representation of knowledge via tuples is a general mechanism for representing, not only networks, but rules and relational tables as well. These representations, however, are beyond the scope of this report.

3. A USER'S GUIDE FOR THE NETWORK EDITOR

This section is divided into three subsections: a brief introduction/orientation to the editor, a sample session illustrating the basic features of the editor, and a reference guide detailing all the options available to the user.

3.1. Four Most Commonly Asked Questions about the Network Editor

What is it?

The network editor is a menu-driven interactive program with modest use of graphics, which runs on a Sun-2 Workstation. The interface is written on top of the SunWindow package developed by Sun Microsystems, Inc. Naturally, being menu-driven, it is easy to figure out how it works just by pressing the right buttons. Experimenting with the editor, and following, the sample session below are the best ways to learn to use the editor. Besieds the sample session and the brief introduction below, the rest of the user's guide is intended only for reference.

How do you display a network?

In displaying a network, a non-graphic approach was taken to allow minimum modification for running the program on machines without graphic capabilities.. Nevertheless, the network structure is readily apparent as the figures in the next section demonstrate. Nodes are represented in boldface. Arcs under nodes are not in boldface, and are set one line below and indented from the main node. Subnodes under arcs are placed similarly under the arc. Additional subnodes are placed on the same line immediately following the preceding subnode.

What kind of things can I do to a network?

At each node or arc in the network, two sets of options are available. One set of options effects the node or arc itself, the other set effects the environment around the node or arc. An example of an option in the first set is changing a printname of a node. An example of an option in the second set is adding an attribute after a node. The first set of options are available via the middle mouse button, the second set via the right mouse button.

How can I determine the portions of the network I want displayed?

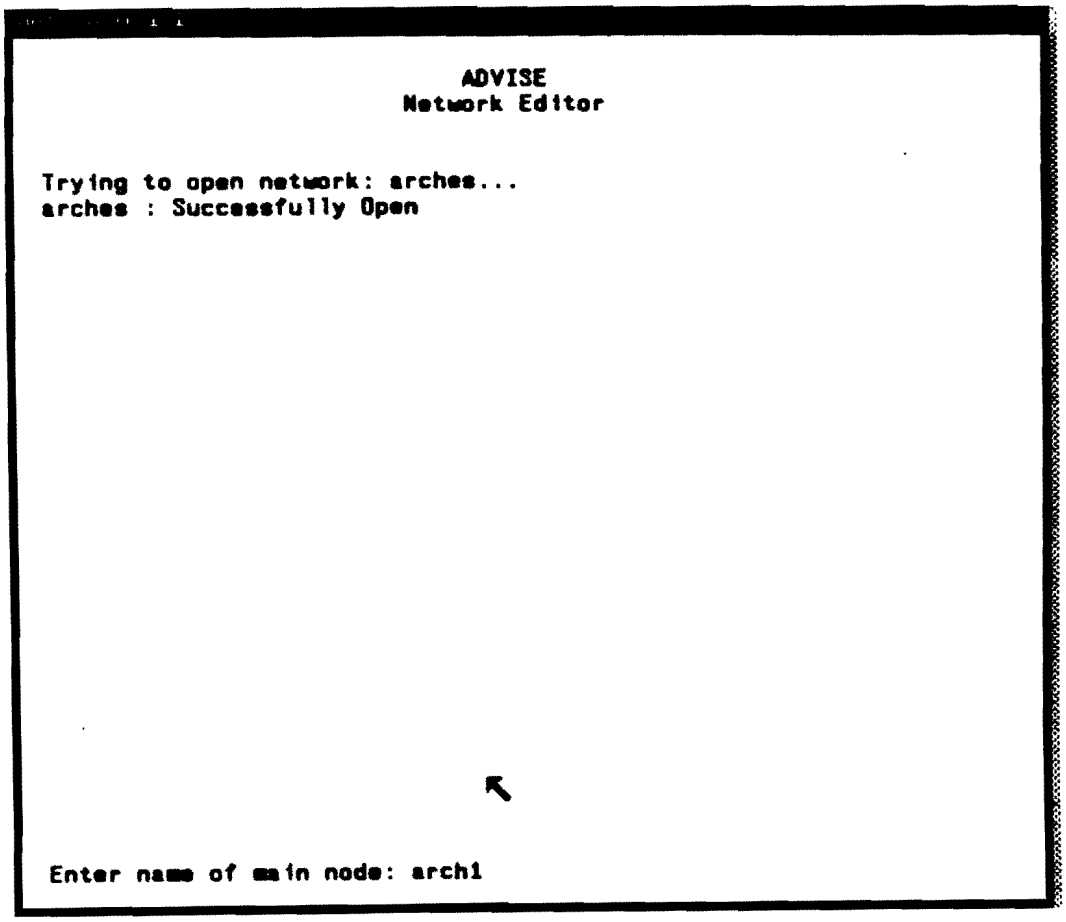
The only other thing a user need know in order to use the network editor is how it clips the network to fit on a display. There are three parameters which affect the display – namely, depth, arc breadth and tuple breadth. Depth is the number of arcs down from the main node to display. Arc breadth is the number of arcs to traverse from each node. And tuple breadth is the number of subnodes to display under the head node. How these parameters affect the display will be seen in the next section.

If in spite of these parameters the network still cannot fit on the screen, the network editor leaves markers that indicate information has been clipped from display. At the top level of the network, these markers are arrows that point in the direction of the missing information. Menus are available at these markers to scroll the top level of the network in order to see the missing information. Beyond the top level of the display, missing information is indicated by a string of dots – "...". Missing information at this level can usually be viewed only by descending the network to make this level the new top level and, if necessary, scrolling or changing the appropriate parameters.

3.2. A Simple Sample Session

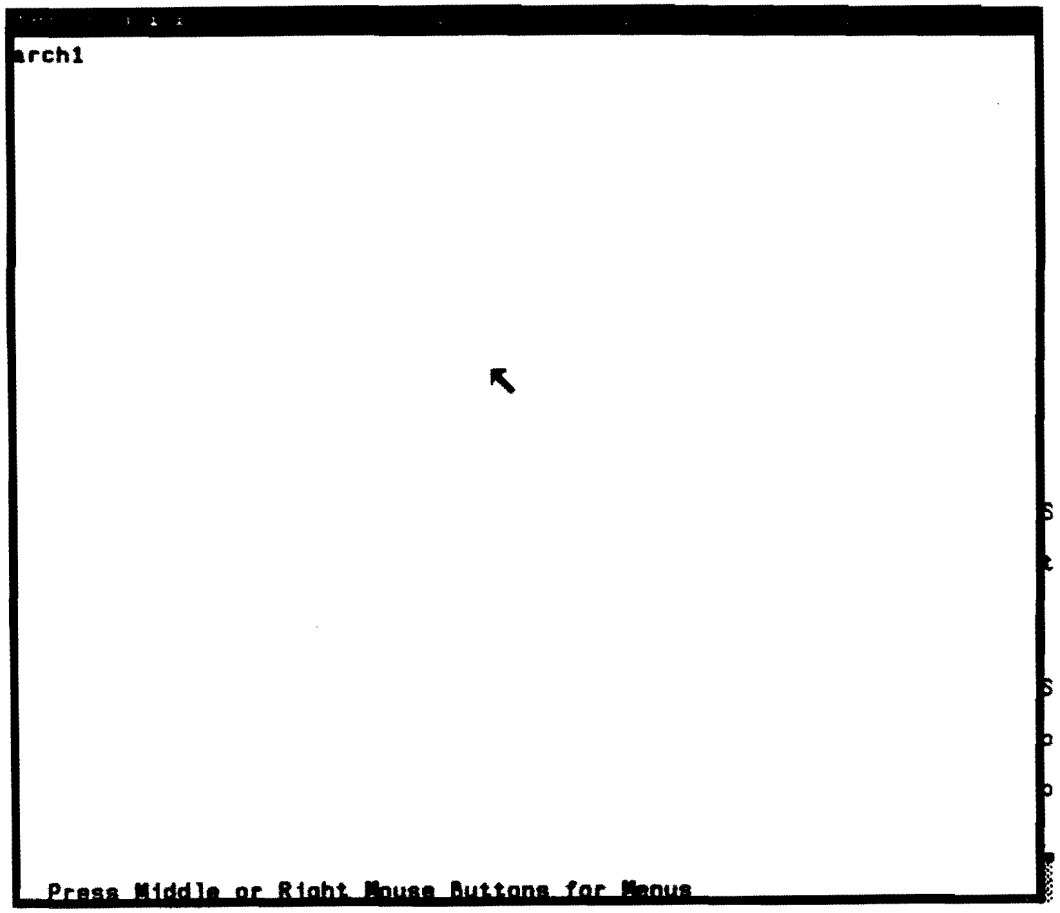
In this section, a simple network is constructed using the network editor. The example is chosen to illustrate the features of a network editor, and is not intended to have any semantics in the context of ADVISE, therefore any resemblance of the network to ADVISE systems, living or dead, is purely coincidental. The network represents a arch made of building blocks.

Here the network editor has just been invoked with the network name "arches" as an argument. A node to be taken as the root node for the display has just been typed in. The editor must be supplied with a network name and a root node before it can begin a session. The network name can be passed as argument or typed in when the editor starts up.

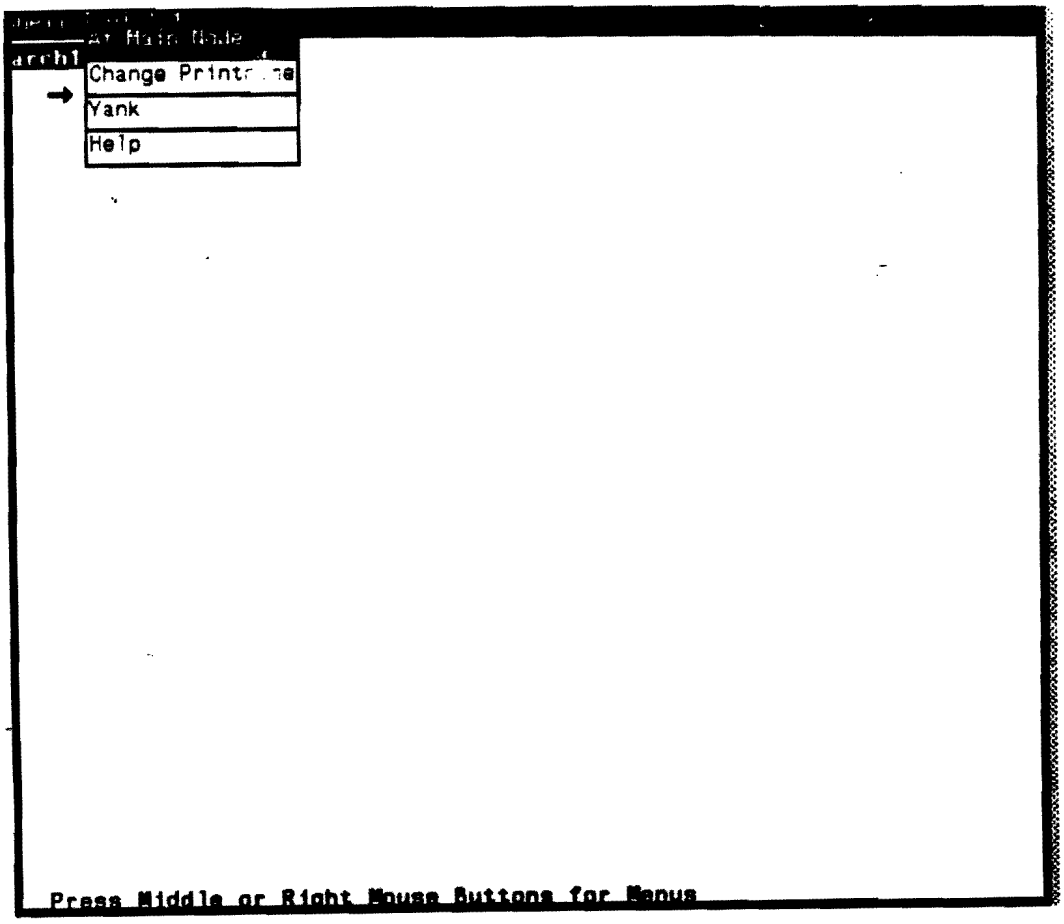


Since there are no other nodes in the network, "arch1" is created and placed at the root of the network. If "arch1" already existed in the network, the structure under this node would be displayed under the default parameters. The default parameters are set to not effect the display, i.e. the window size is the only limiting factor to the display.

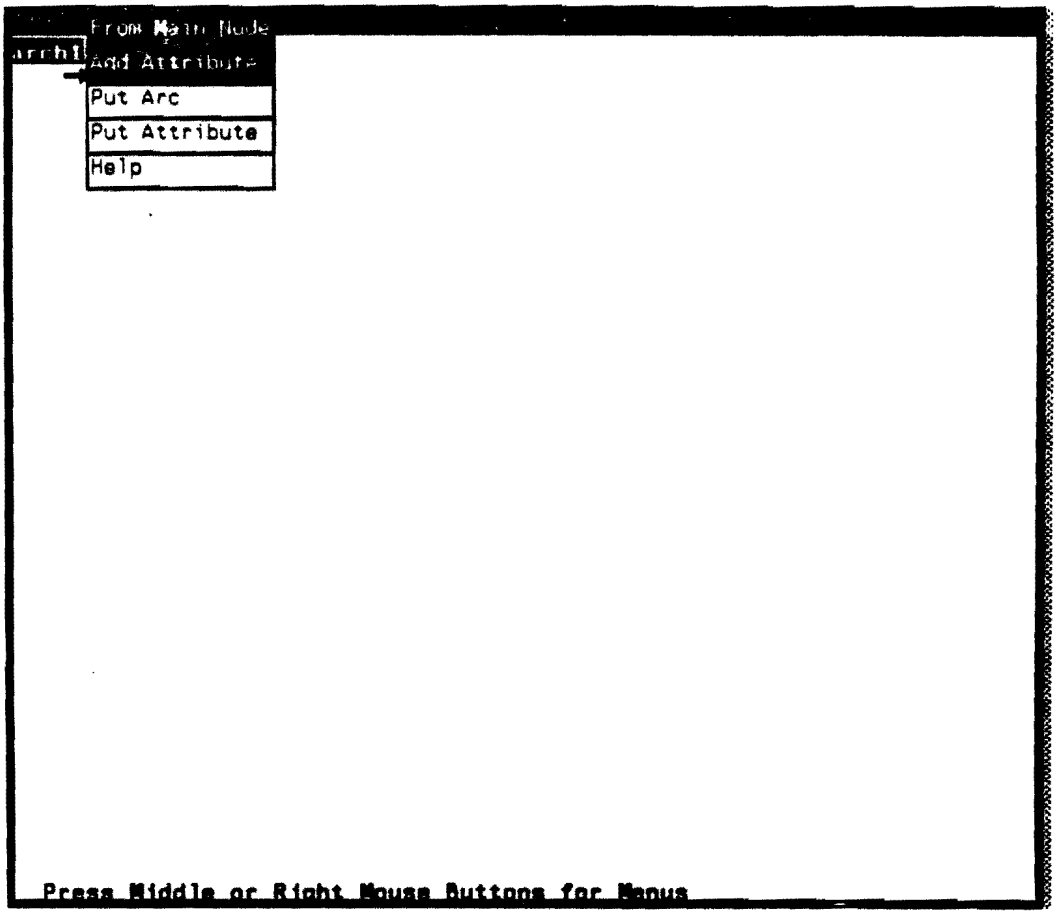
A prompt for menus appears at the bottom of the screen.



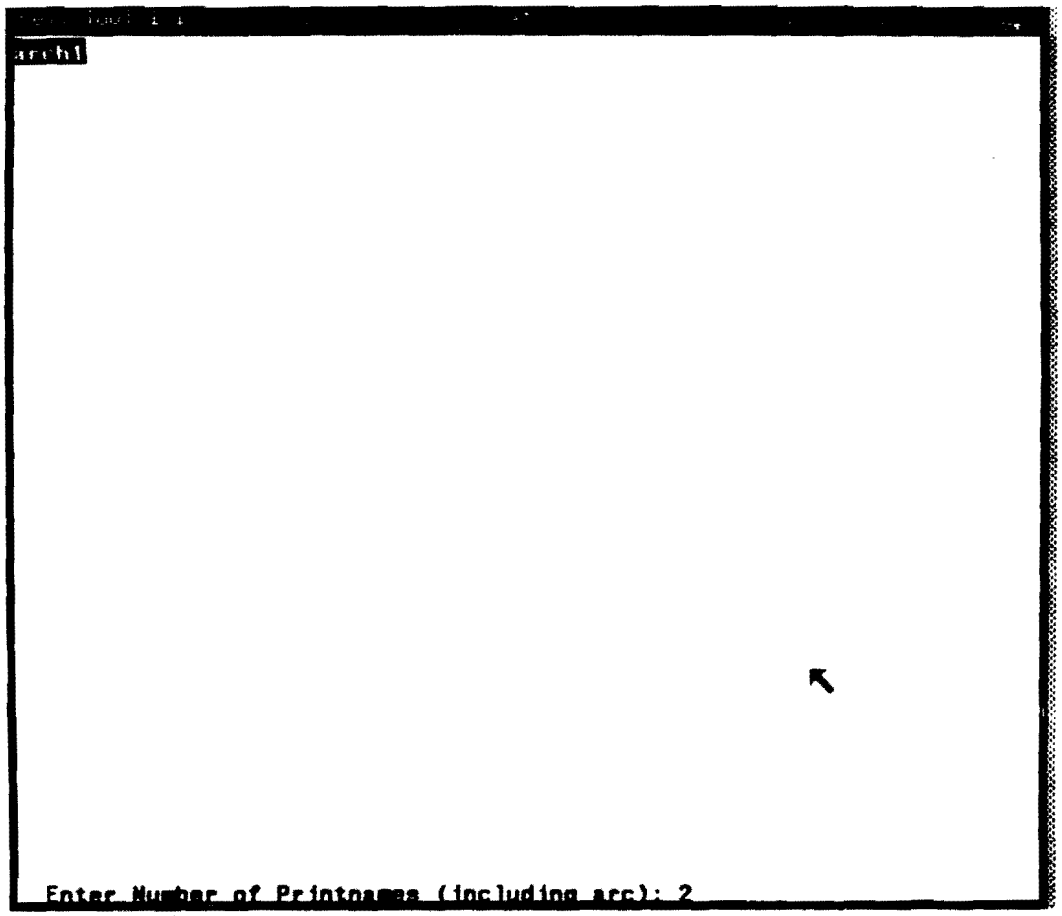
The user moves the mouse to the "arch1" node, which becomes highlighted. Then the user presses the right button on the mouse and the menu for "local" options appears. "Yank" places the node in a buffer to be added into a tuple at a later date. Help is available with all menus.



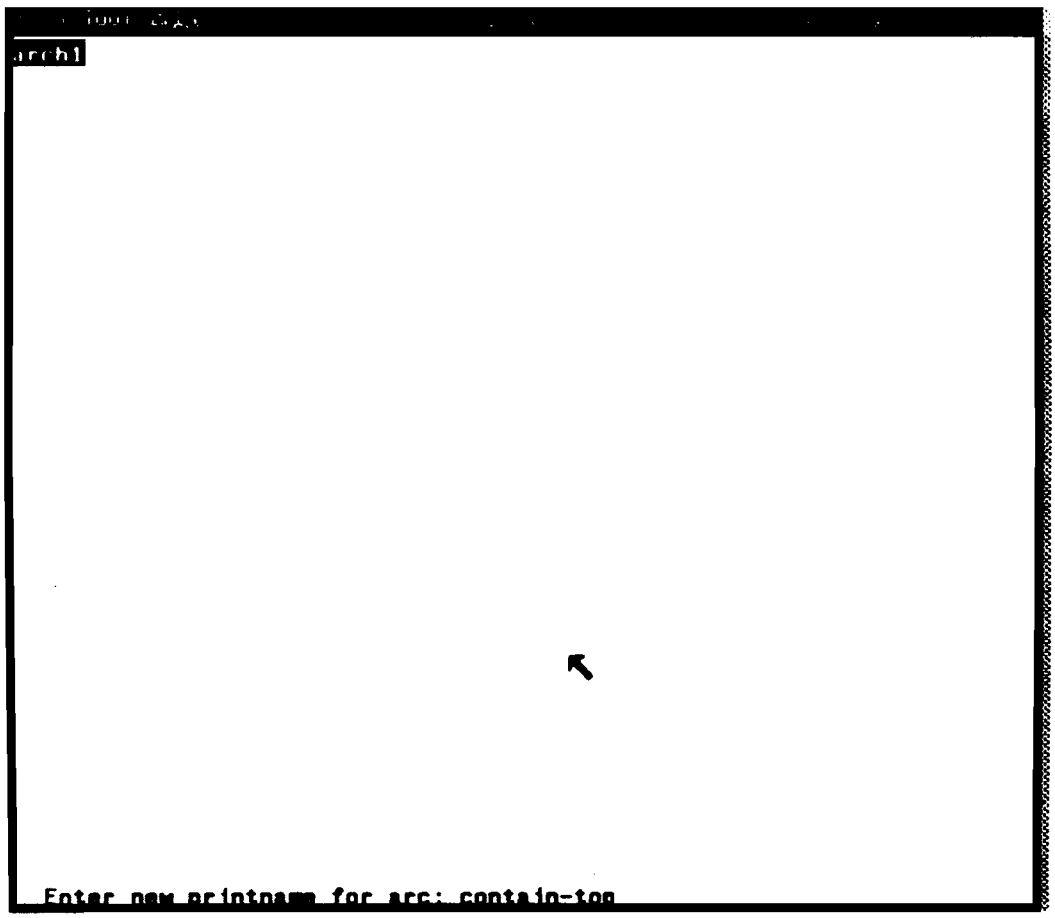
Pressing the middle button while at the node reveals the desired option of adding an attribute (slot).



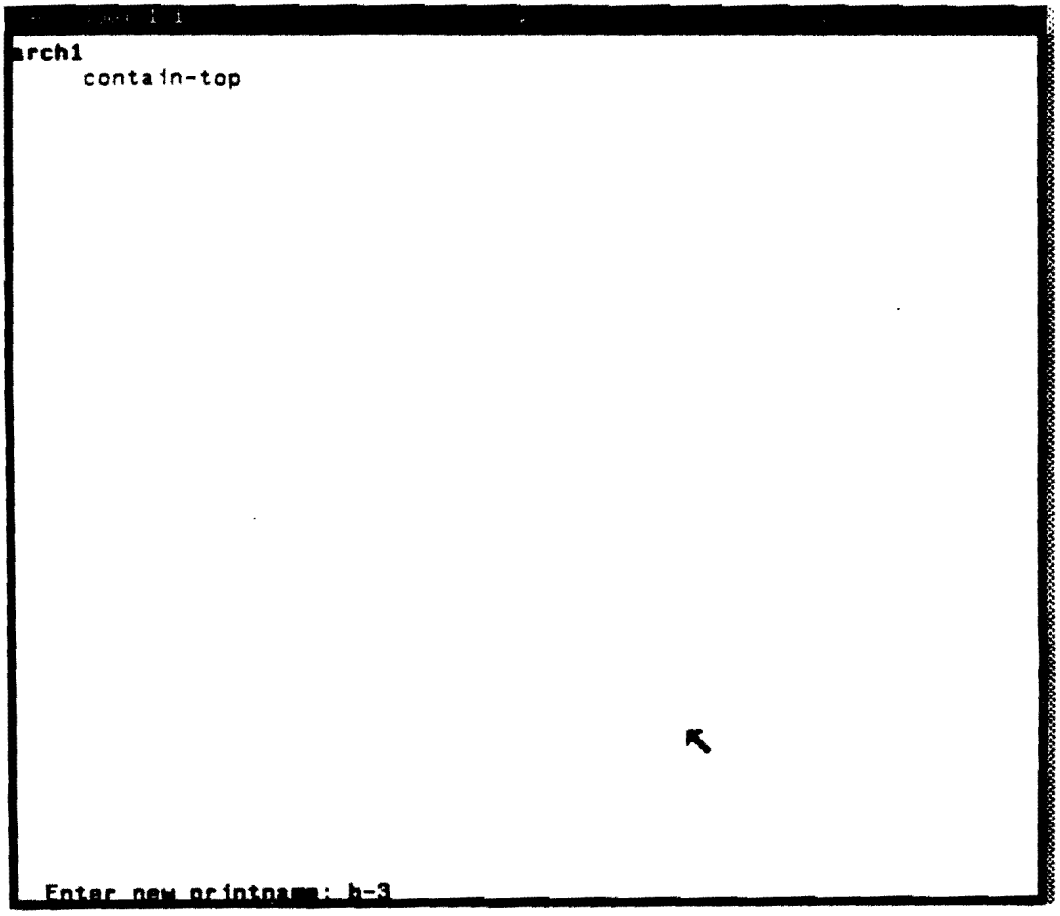
Selecting "Add Attribute" brings up a prompt for the number of nodes (printnames) including the arc in the tuple which the network editor will add into the network.



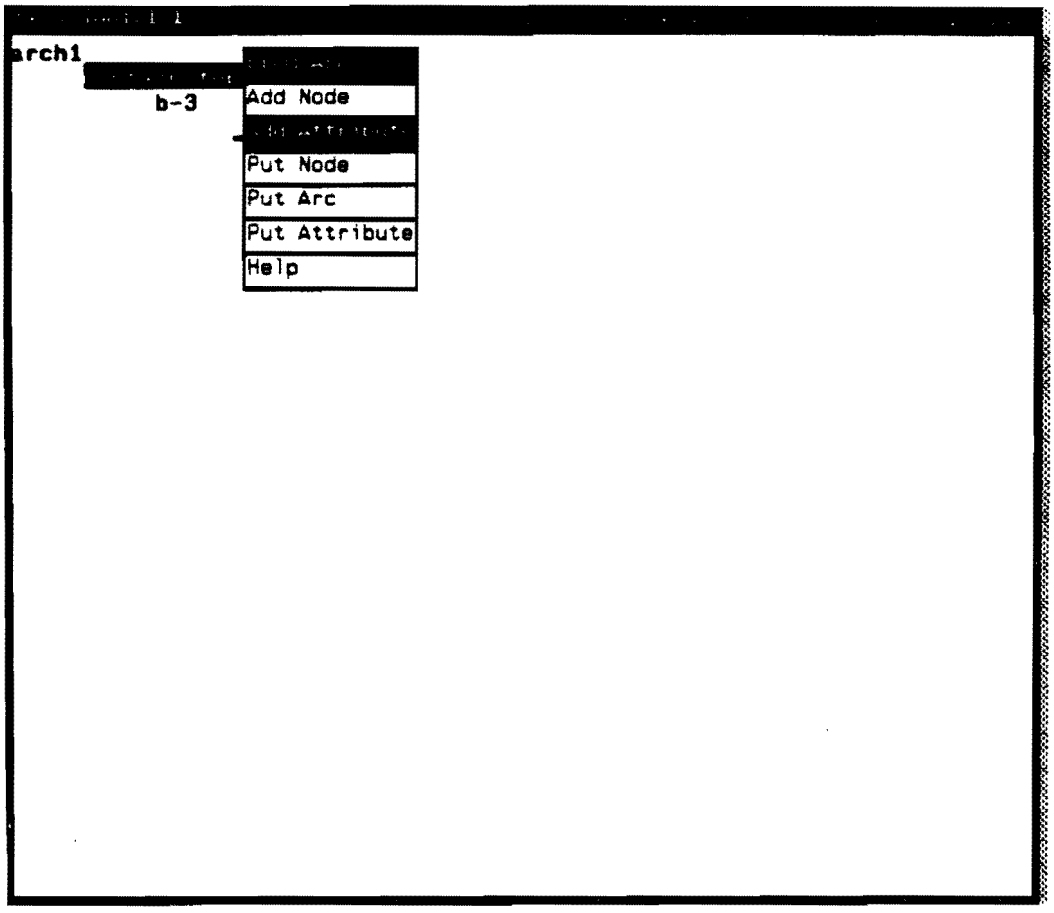
Next the user is prompted for the printname of the arc. The user types in this name.



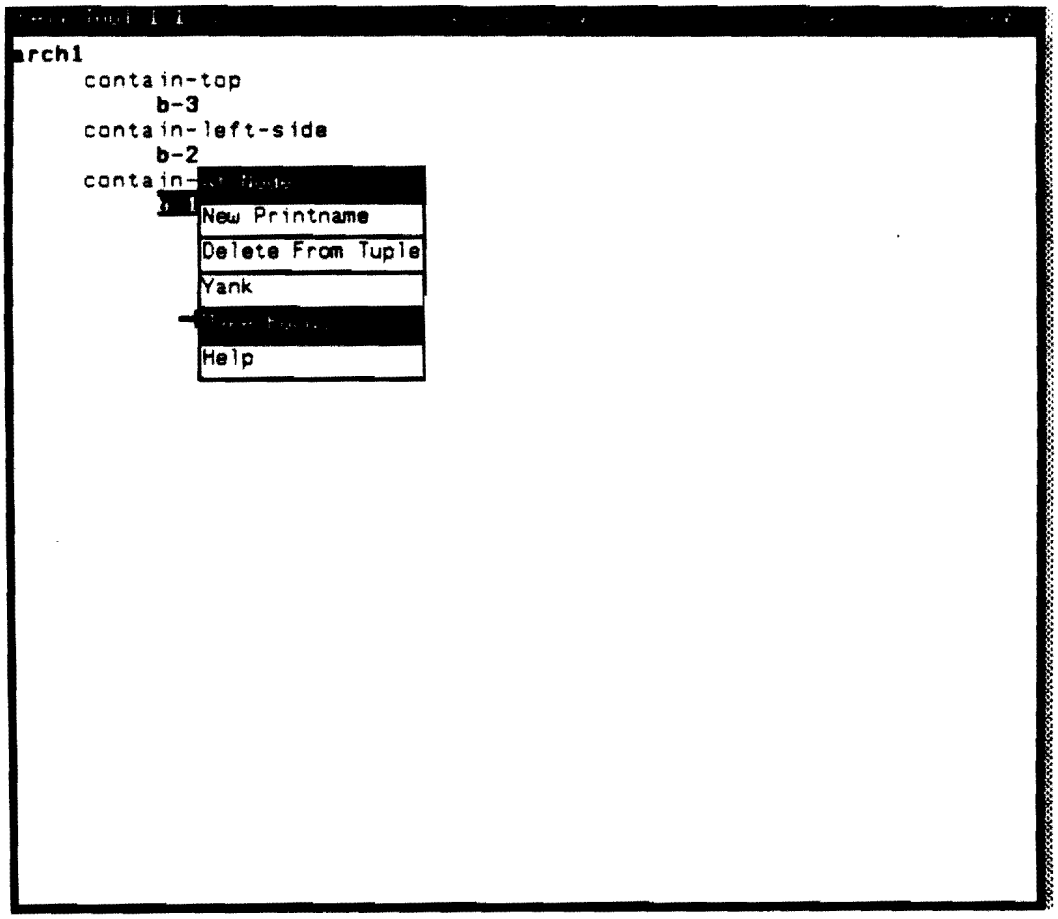
When the users presses return, the arc appears in the network and the name of the node is solicited. The users types the node name.



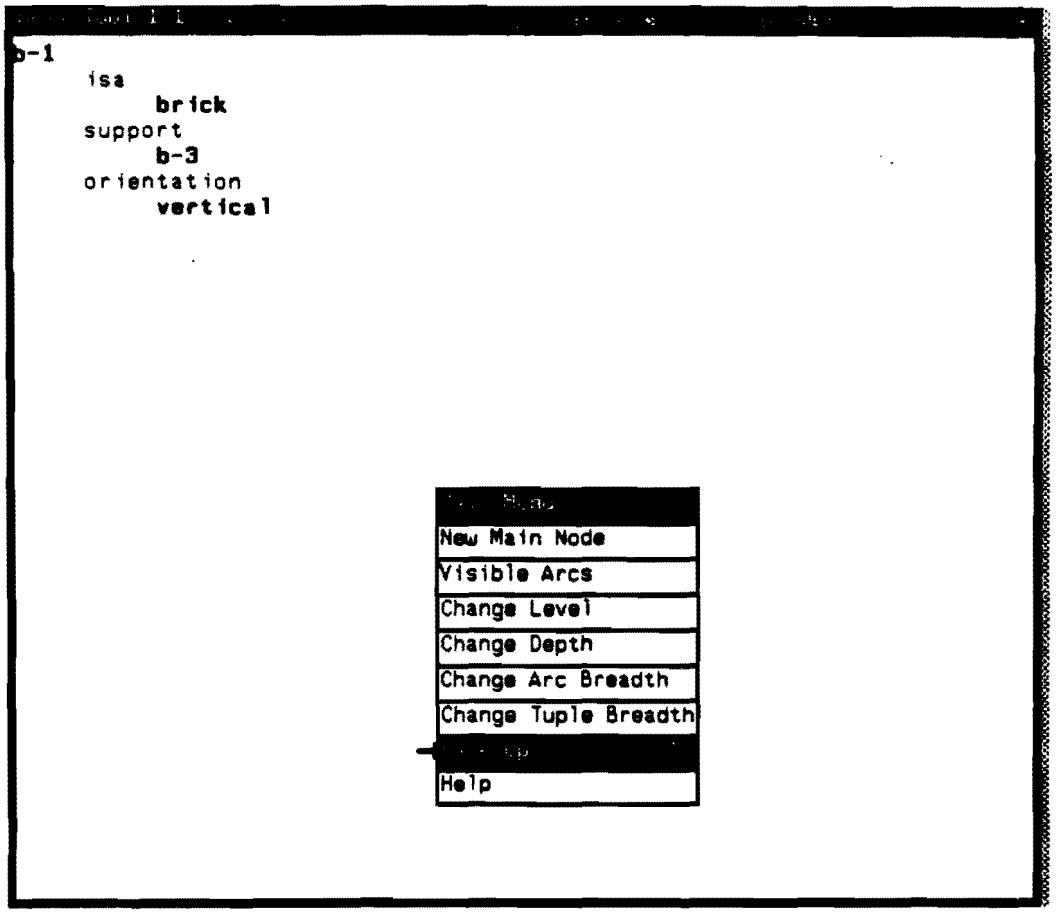
The entire attribute now appears in the network. No more prompt appears. The user now moves to the arc to add the next attribute below the first attribute.



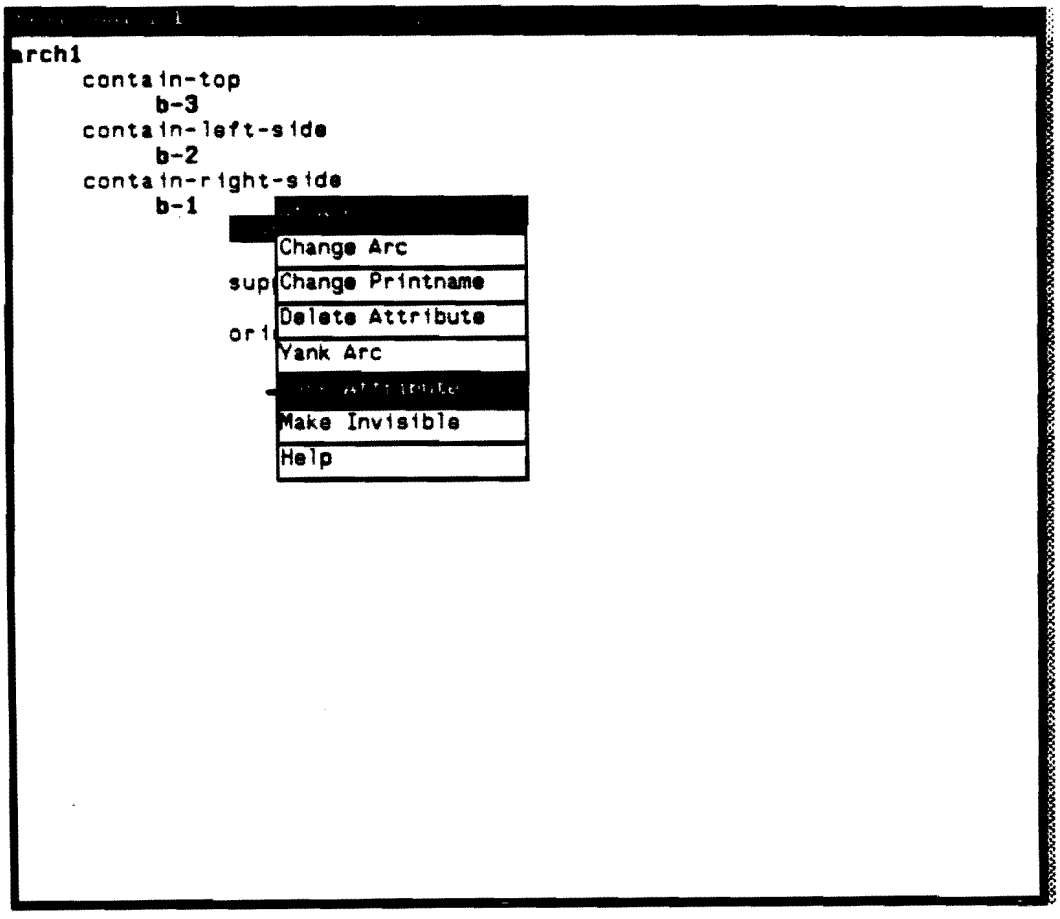
Continuing as with the first attribute the user has added all the arcs under "arch1". He now wishes to add arcs under the "b-1" node. To do so he must first make b-1 the new focus node.



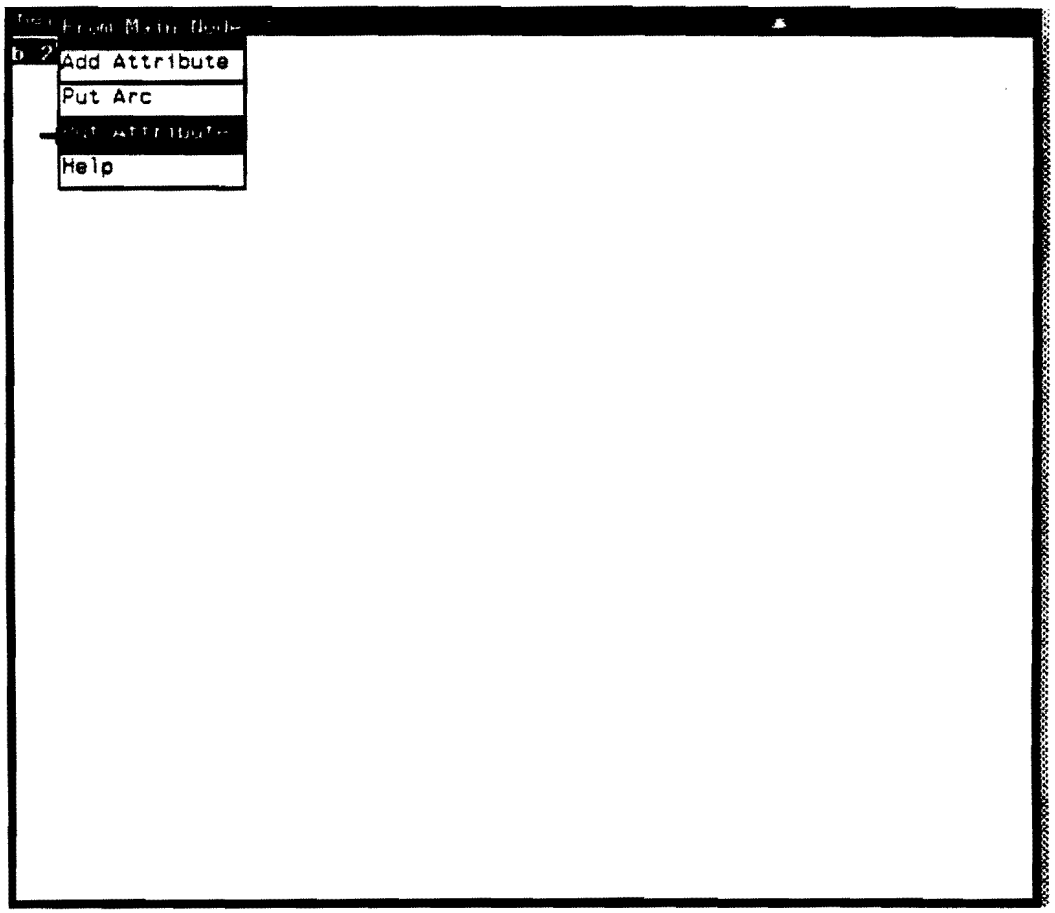
"b-1" becomes the new focus node. The user has used "Add Attribute" as before to add all the appropriate nodes and is now ready to "Back Up" to the previous root node. "Back Up" is an option from one of two menus that aren't associated with any node. The other is a global menu with options such as editing a new file, writing this file, quitting, etc.



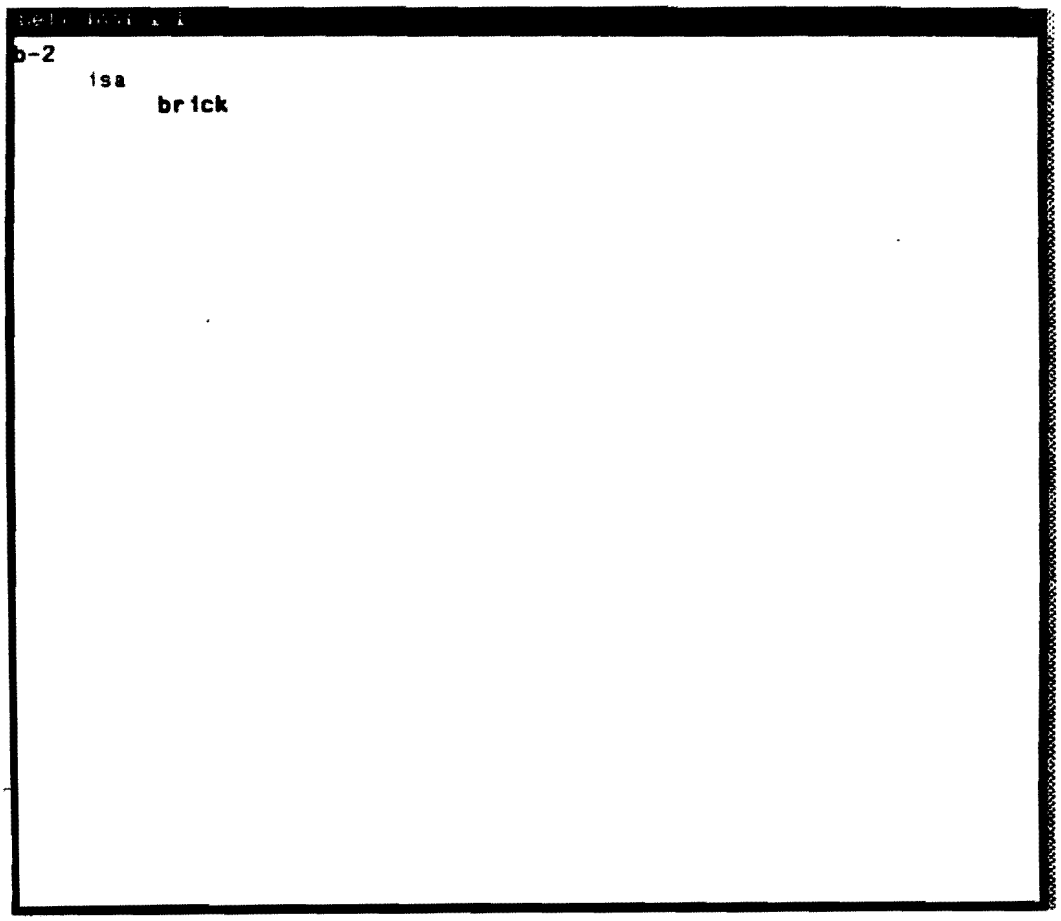
Next the user wants to add attributes under "b-2". To do so he must make b-2 the focus as he did before. This time however, he notes that he wants to add some of the attributes that are already under b-1. He chooses to "Yank" the "isa" attribute.



After making "b-2" the new focus, the user wishes to "Put" the attribute he just "Yanked" under "b-2".

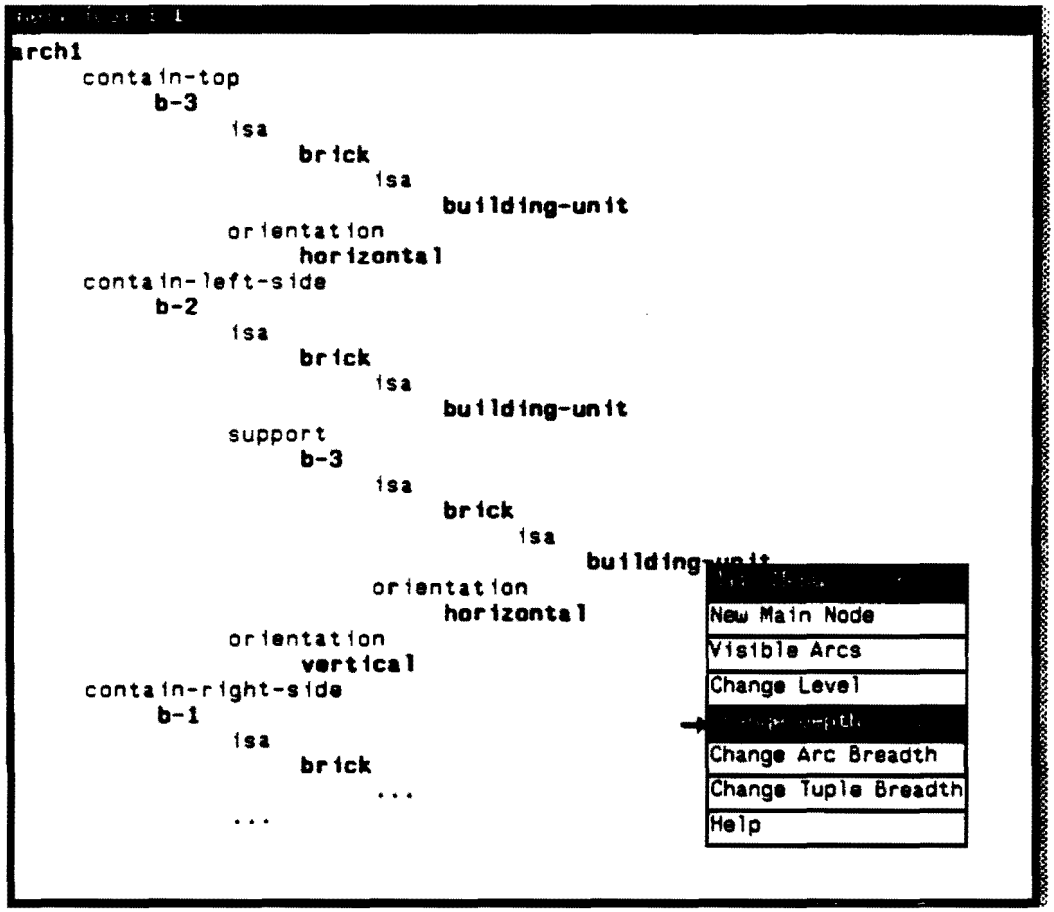


The "Put" succeeds. The user continues adding, yanking and putting until the entire network is complete.



Now the network for "arch1" is complete. Note the three dots near the bottom of the screen. These are markers that indicate information would appear beginning at that position.

At this point the user may enter a new root node, such as "arch2", and create a network from there or he may choose to alter the display by changing depth, for example.

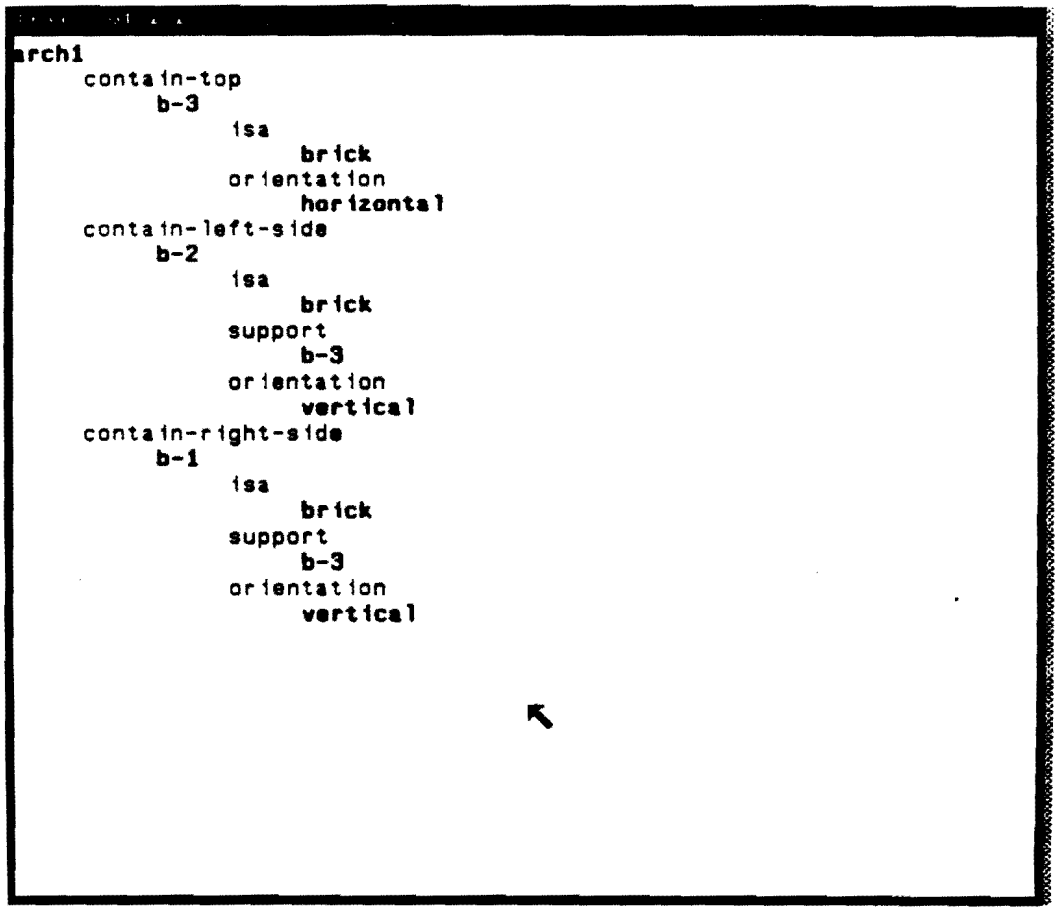


The user indicates he wishes the depth changed from an "unlimited" default to 2.

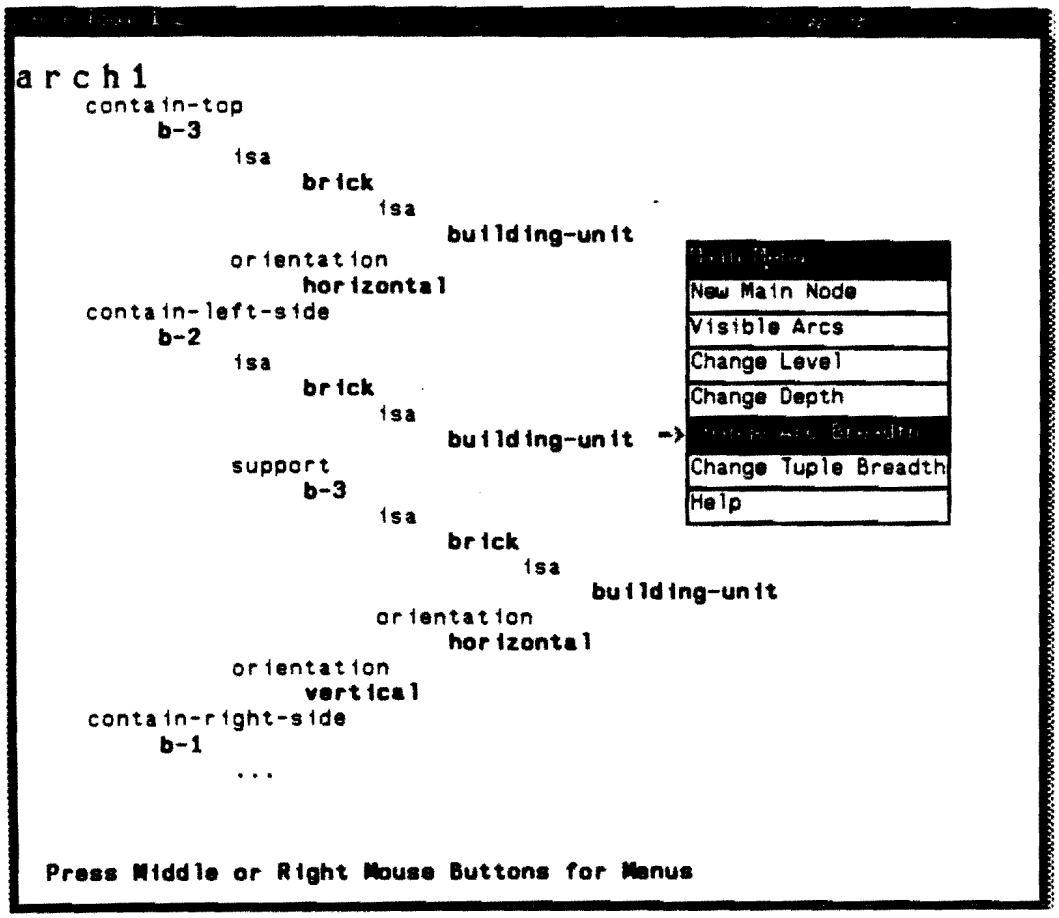
```
arch1
    contain-top
        b-3
            isa
                brick
                    isa
                        building-unit
            orientation
                horizontal
    contain-left-side
        b-2
            isa
                brick
                    isa
                        building-unit
            support
                b-3
                    isa
                        brick
                            isa
                                building-unit
                    orientation
                        horizontal
            orientation
                vertical
    contain-right-side
        b-1
            isa
                brick
                    ...
            ...

Current Depth is  99
New Display Depth (Enter a number): 2
```

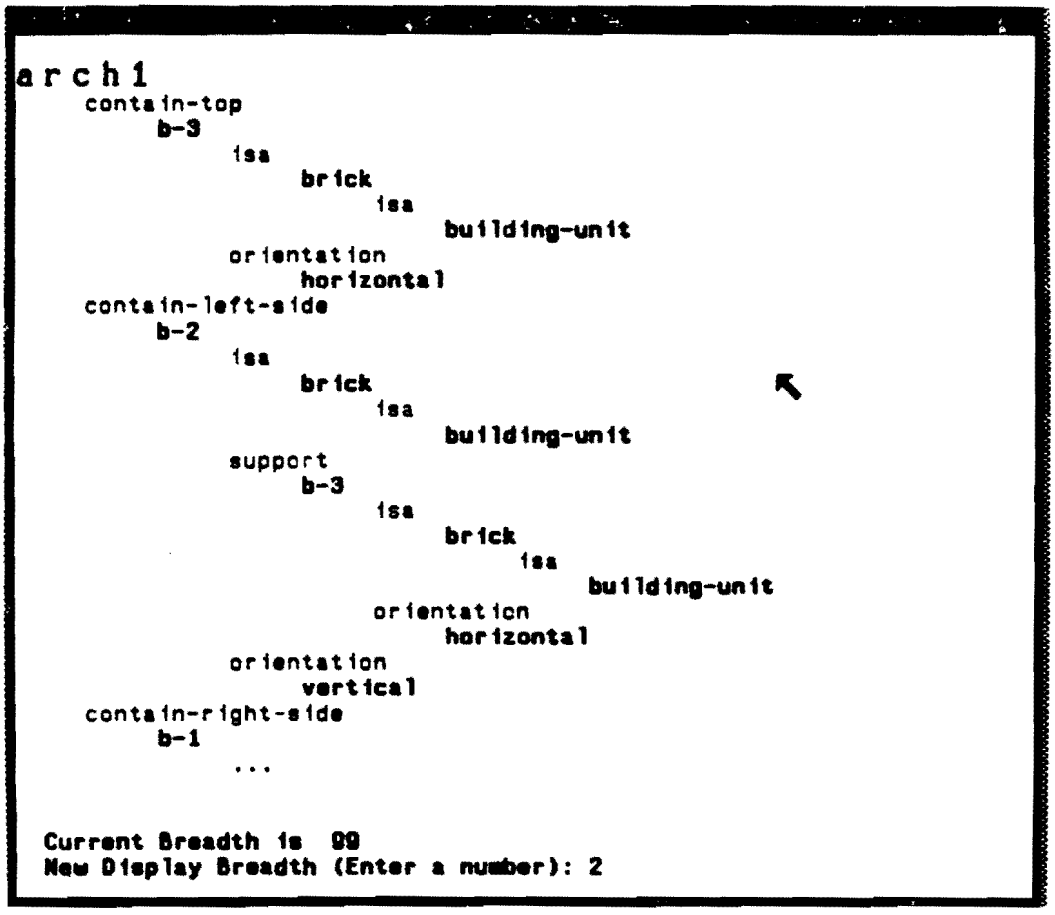
Now the depth has been changed to 2. Note that the markers indicating clipped information have disappeared.



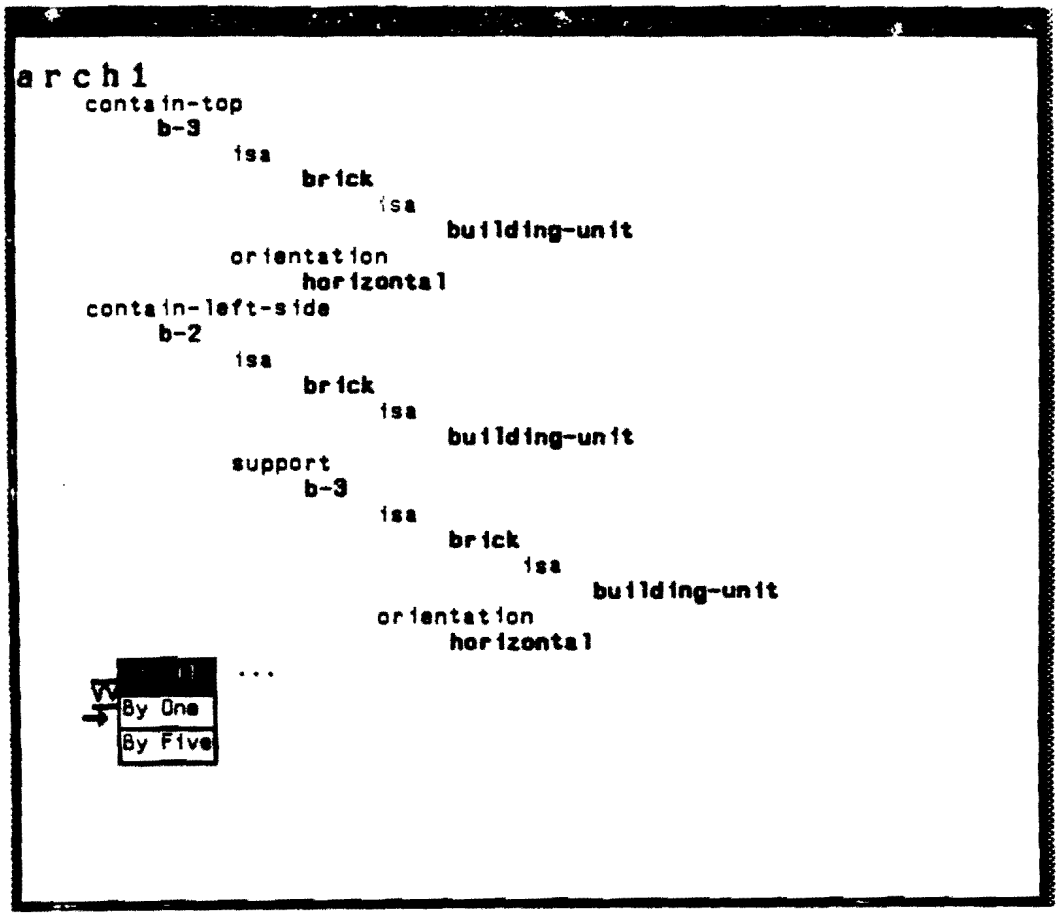
Instead of limiting the depth, the user may have chose to limit the breadth of the network instead.



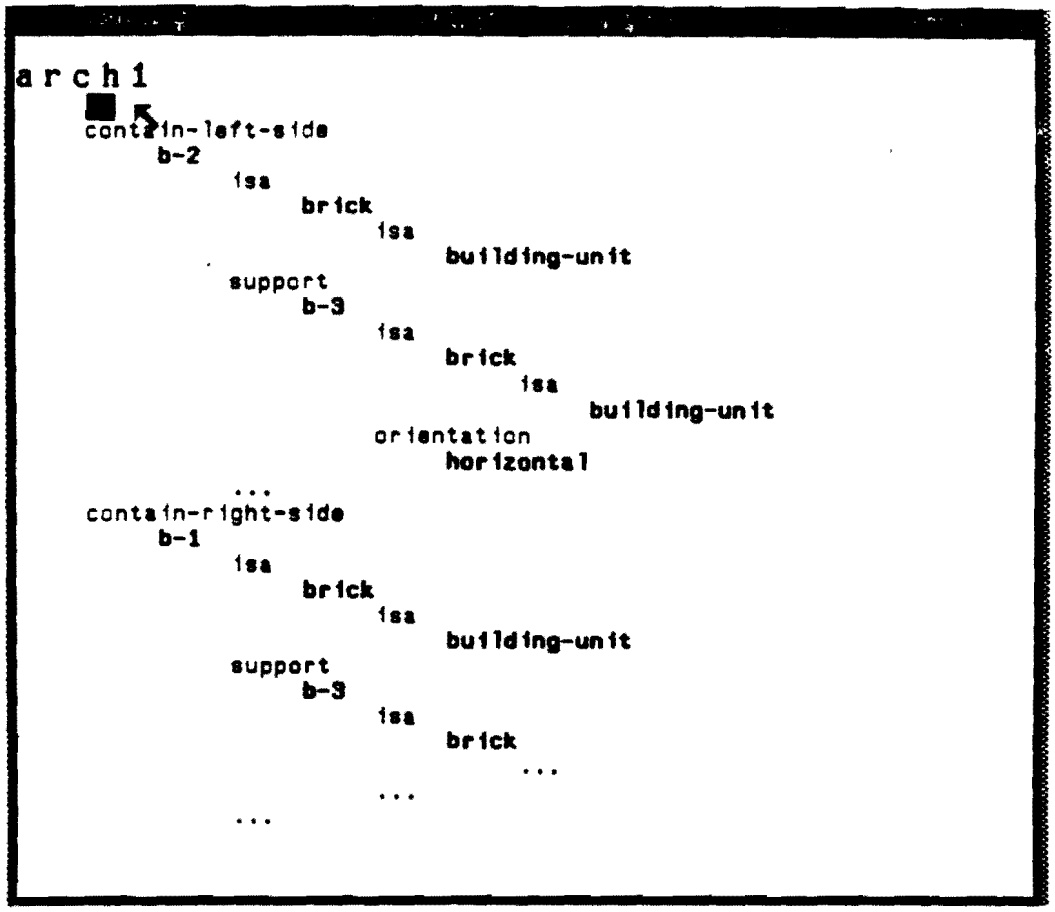
The user indicates he wishes the breadth changed from an "unlimited" default to 2.



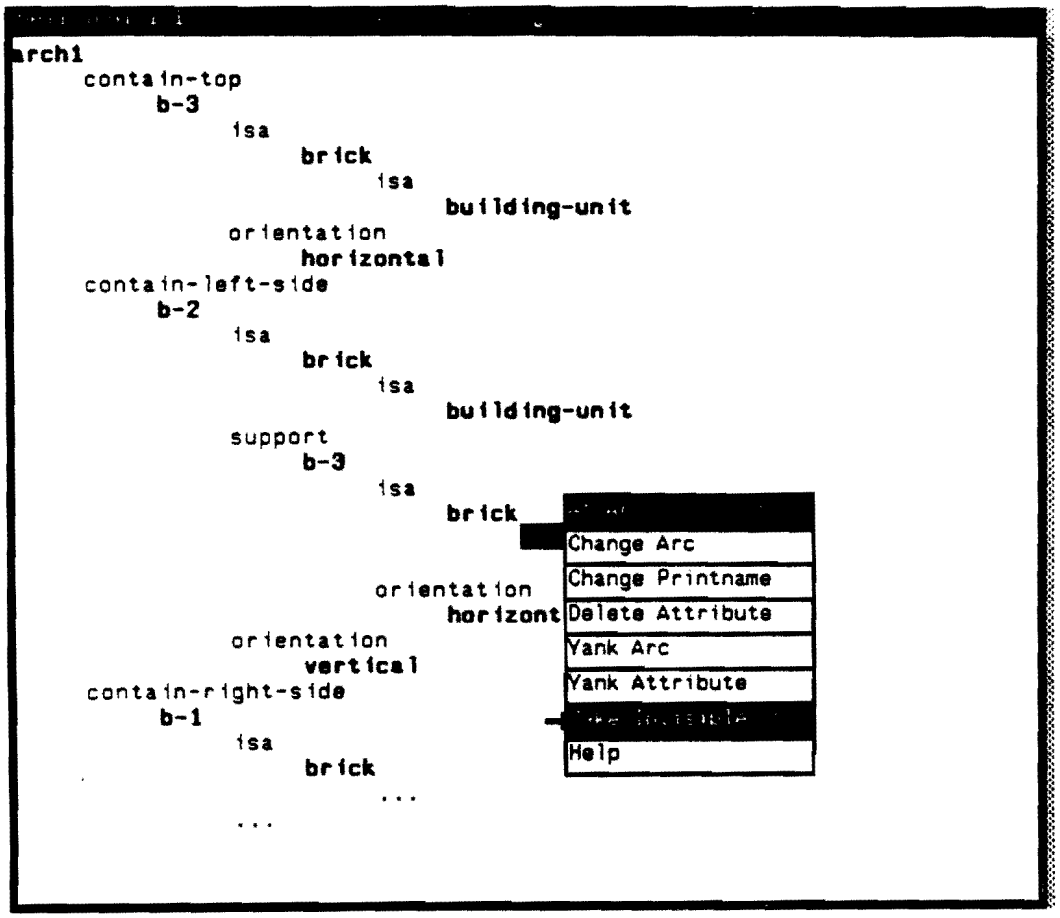
Now the breadth has been changed to 2. Downward-pointing arrows have appeared at the bottom of the screen indicating that an arc has been clipped from the screen. The user moves to these arrows and presses either the middle or right button for the scroll menu.



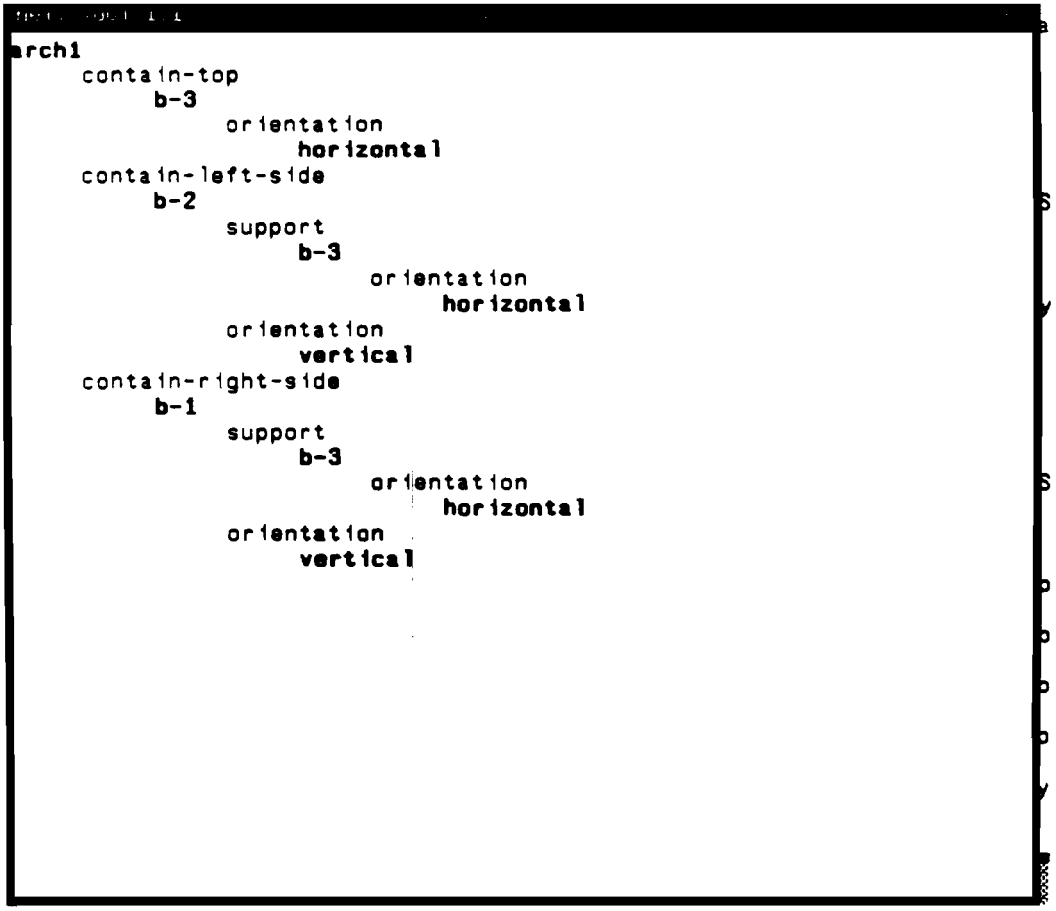
Now the first arc under "arch1" has been clipped so that the last two arcs are visible. Upward pointing arrows allow the user to scroll upwards as before. Left and right arrows provide analogous left-right scrolling when the subnodes of the main node extend beyond the edge of the screen or when the tuple breadth prevents all the subnodes from being seen.



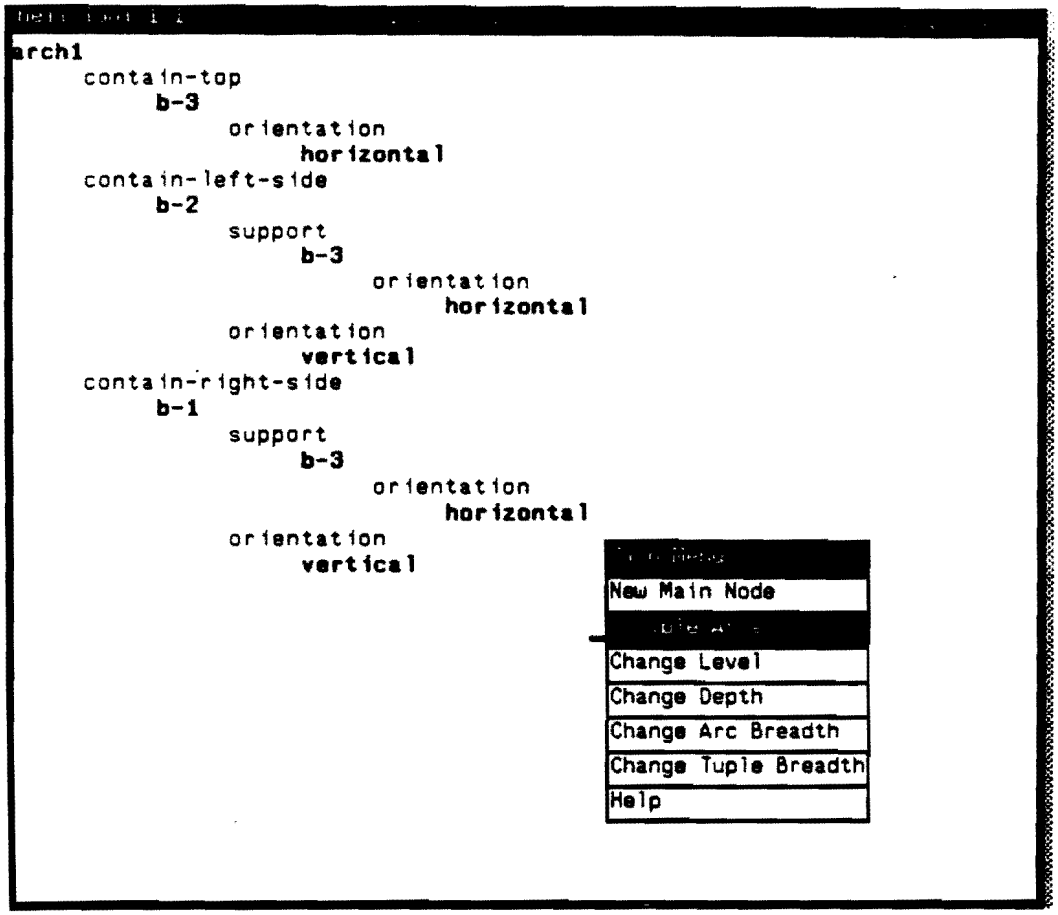
Another way to alter the display is by making arcs invisible. With the breadth and depth reset to their default values, the user causes "isa" links to disappear from the display.



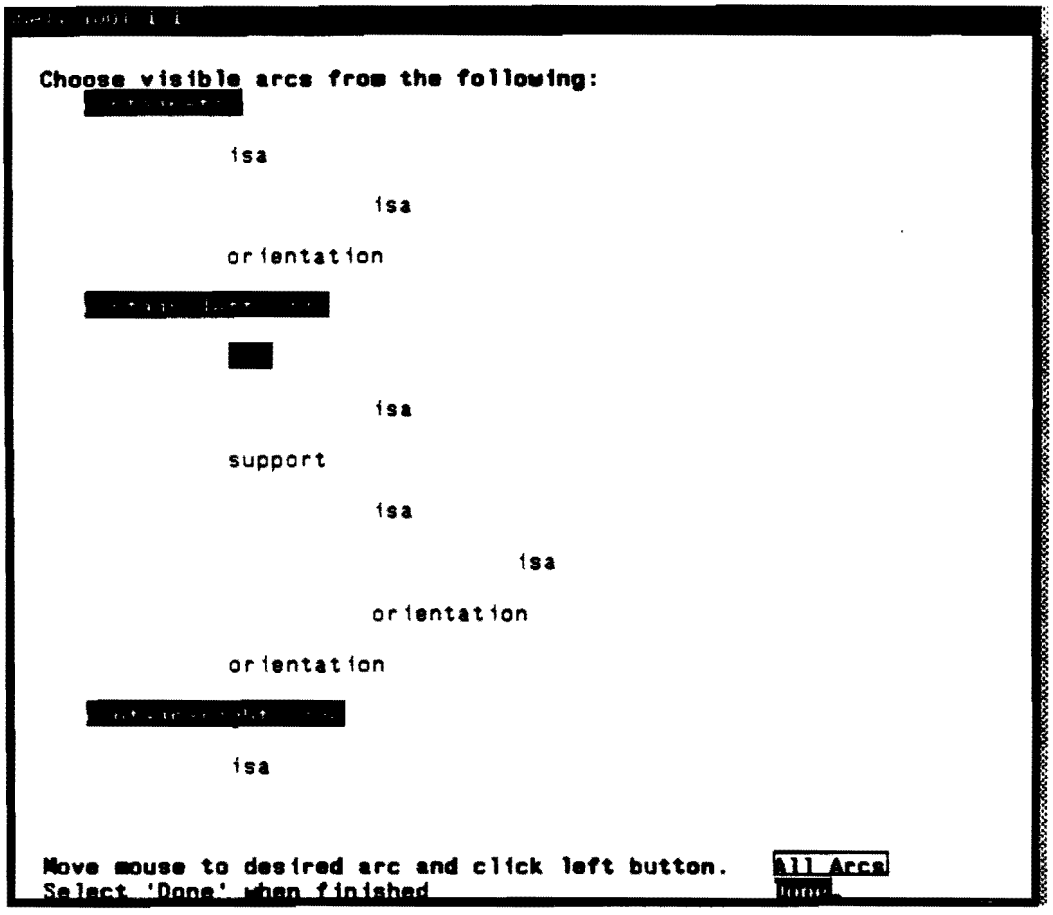
This action, as did the change depth option, also caused the clipping markers to disappear.



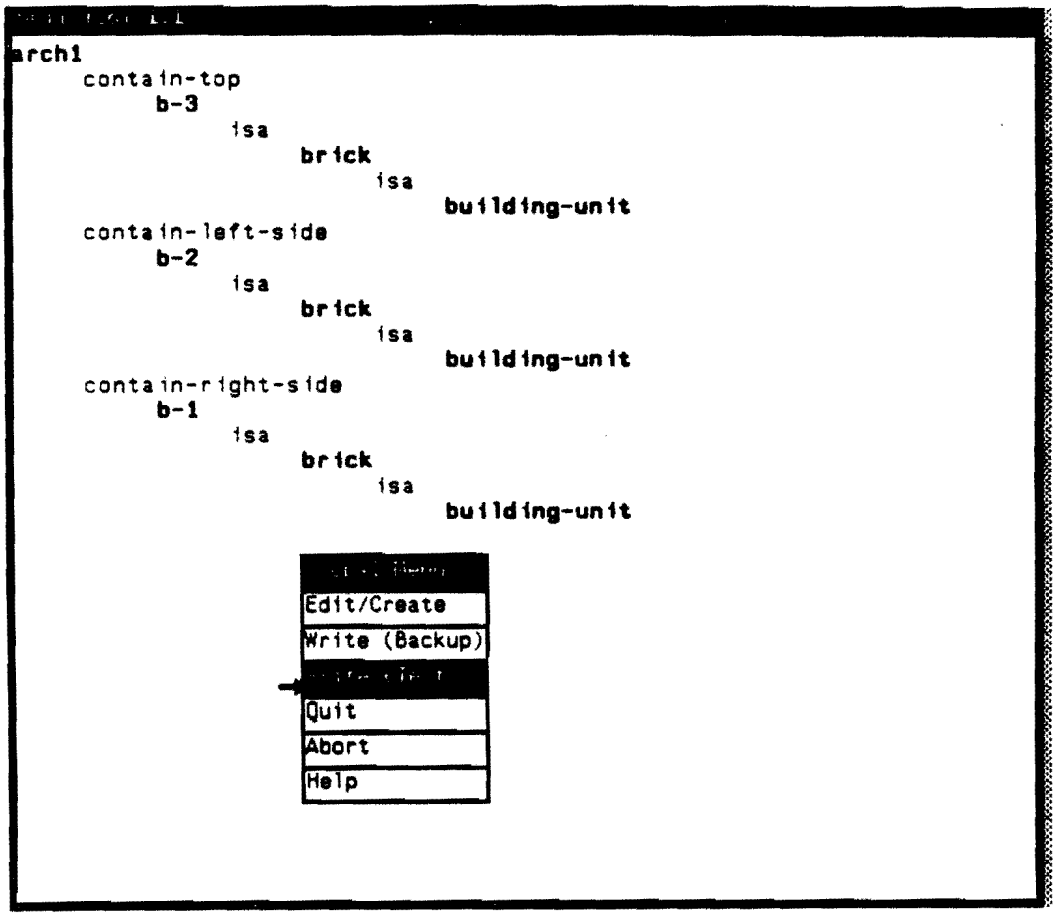
Still another way to limit the display is by specifying a subset of the visible arcs as the only ones to be displayed.



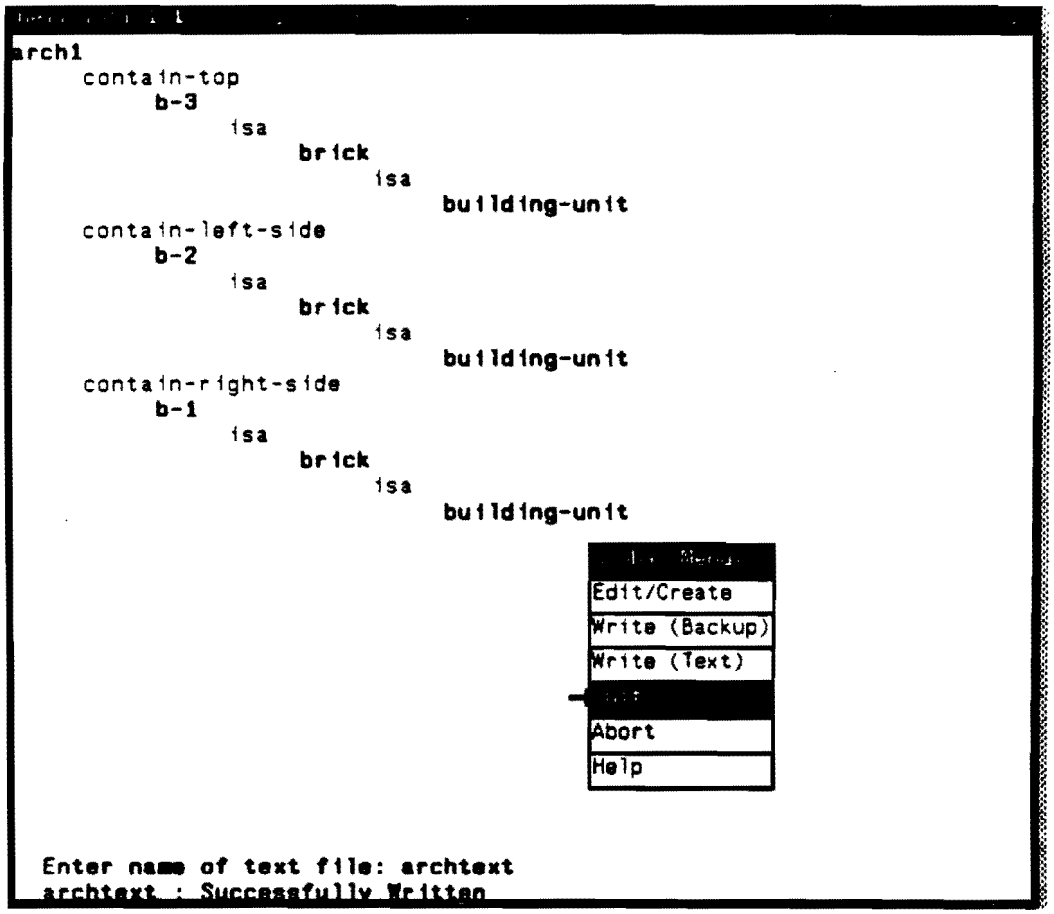
Here the user has selected the "contains" arcs and the "isa" arcs to be the only ones visible. This will allow him to see inheritance relationships.



After seeing the inheritance relationships via is-a links, the user decides he wants to end the session. He presses the middle mouse menu for the the global option menu. He chooses "Write (Text)" because he (and we) want to see the tuple representation of his network.



The user types in a file name for the text file. The system notifies him of success (or failure). Now the user "Quits". The actual network is written to file given at the start of the program.



```
(arch1 (
    (contain-top b-3 )
    (contain-left-side b-2 )
    (contain-right-side b-1 ) ))
(contain-top ( ))
(b-3 (
    (isa brick )
    (orientation horizontal ) ))
(contain-left-side ( ))
(b-2 (
    (isa brick )
    (support b-3 )
    (orientation vertical ) ))
(contain-right-side ( ))
(b-1 (
    (isa brick )
    (support b-3 )
    (orientation vertical ) ))
(isa ( ))
(brick (
    (isa building-unit ) ))
(support ( ))
(orientation ( ))
(vertical ( ))
(horizontal ( ))
(building-unit ( ))
```

The textual representation of tuples for "arches"

3.3. Network Editor Reference Guide

This section gives descriptions of all the options available from each menu.

3.3.1. Menus Available from the Main Node

At Main Node via Right Mouse Button

Change Printname	Allows you to change the printname of this node EVERYWHERE it occurs in the network
Yank	Allows you to yank this node into the node buffer for subsequent 'Put Node' operations
Enter Dictionary	If the node is not in the dictionary, you may enter it there if this option is present. If and only if a node is in the dictionary, it can be made the main node via the 'New Main Node' option on the Main Menu
Help	Prints this message

At Main Node via Middle Button

Add Attribute	Allows an attribute (an arc followed by zero to 254 nodes) to be added under the main node.
Put Arc	The contents of the arc buffer will be put under the main node as the first attribute.
Put Attribute	The contents of the attribute buffer will be put under the main node as the first attribute.
Help	This message is displayed.

3.3.2. Menus Available at Nodes other than the Main Node

At Nodes with Numeric Values via Right Mouse Button

Change Node	Allows you to change the number to a new number or to a new node.
Delete From Tuple	This node is removed from its current tuple position.
Yank	Allows you to yank this node into the node buffer for subsequent 'Put Node' operations
Help	Prints this message

At Nodes with Non-Numeric Values via Right Mouse Button

Change Printname	Allows you to change the printname of this node EVERYWHERE it occurs in the network
Delete From Tuple	This node is removed from its current position. All other occurrences of the node remain intact.
Yank	Allows you to yank this node into the node buffer for subsequent 'Put Node' operations
Make Focus	This node becomes the new top node in the display of the network.
Help	Prints this message

At Nodes with Non-Numeric Values via Middle Mouse Button

Add Node	Allows a node to be added to the right of the current node in the attribute in which the current node occurs.
Put Node	The contents of the node buffer will be put to the right of the current node in the attribute in which the current node occurs.
Help	This message is displayed.

3.3.3. Menus Available from Arcs

At Any Arc via Right Mouse Button

Change Arc	Allows a new arc to be inserted in place of the arc in the current attribute.
Change Printname	Allows the printname of the current arc to be changed EVERYWHERE it occurs in the network.
Delete Attribute	Removes this arc and all subnodes from underneath the head node, i.e. the entire tuple is removed.
Yank Arc	Places this arc into the arc buffer to be used in subsequent Put Arc operations
Yank Attribute	Places the attribute (this arc and all nodes underneath it) into the attribute buffer to be used in subsequent Put Attribute operations.
Make Invisible	Inhibits the display of this arc and all other occurrences of this arc as well as everything underneath them.
Enter Dictionary	If an arc, does not appear in the dictionary, this option allows you to put it there.
Help	Displays this message.

From Arcs Under the Main Node via Middle Mouse Button

Add Node	Allows a node to be added immediately under the current arc in the attribute in which the current arc occurs. Nodes presently under the arc are shifted to the right.
Add Attribute	Allows an attribute (an arc followed by zero to 254 nodes) to be added under the main node below the attribute which contains the current arc.
Put Node	The contents of the node buffer will be put immediately under the current arc in the attribute in which the current arc occurs. Any nodes presently under the arc are shifted to the right.
Put Arc	The contents of the arc buffer will be put under the main node below the attribute which contains the current arc.
Put Attribute	The contents of the attribute buffer will be put under the main node below the attribute which contains the current arc.
Help	This message is displayed.

At All Other Arcs via Middle Mouse Button

Add Node	Allows a node to be added immediately under the current arc in the attribute in which the current arc occurs. Nodes presently under the arc are shifted to the right.
Put Node	The contents of the node buffer will be put immediately under the current arc in the attribute in which the current arc occurs. Any nodes presently under the arc are shifted to the right.
Help	This message is displayed.

3.3.4. Menus Available When Not at an Arc or a Node

Via Middle Button

Edit/Create	Start a new session with a new network
Write (Backup)	The edited network will be written to the file given at startup
Write (Text)	A text representation of the edited network will be written to a specified file.
Quit	Graceful exit from a session, updating the network
Abort	Immediate exit from editor, no update.
Help	This message

Via Right Button

New Main Node	Allows you to enter the name of a node which will become the new top node in the display.
Visible Arcs	Allows you to select a subset of arcs to follow in displaying the network
Change XXXXX	Allows you to enter a new value for XXXXX - depth, breadth, or tuple breadth - which will effect the display accordingly
Back Up	If present in menu, allows ascension of the network to the previous top node
Help	Prints this message

Via Visible Arcs Option from Right Button Menu Above

See Invisible Arcs	When this option is available, arcs exist which would be displayed had they not been designated invisible. Select this option if you wish to make some of these arcs visible again.
Make New Arcs Visible	If you wish to select a subset of arcs as the only visible arcs from those arcs that are currently visible select this option.

4. CONCLUSION

In section 3, the basic features of the network editor were displayed. Although the example network was naive, it served to illustrate much of the editor's overall usefulness with regard to larger more realistic networks.

An immediate application of the network editor is working with the BABY system. The implementation of the network editor serves as the basis for a similar menu-driven rule editor for editing rule-structured knowledge bases in ADVISE.

5. ACKNOWLEDGEMENTS

The author would like to thank everyone associated with the ADVISE project, particulary Professor A.B. Baskin and Carl Uhrik for their input and support in building this editor.

REFERENCES

[1] Michalski, R.S, et.al. *A Technical Description of the ADVISE Meta-expert System*, Department of Computer Science, University of Illinois, May 5, 1983.

[2] Reinke, R., *PLANT/ds: An Expert System for the Diagnosis of Soybean Diseases Common in Illinois, User's Guide and Program Description*, Department of Computer Science, University of Illinois, October 1983.

[3] Rodewald, R.E., *BABY: An Expert System for Patient Monitoring in a Newborn Intensive Care Unit*, Department of Computer Science, University of Illinois, July 1984.

[4] Spackman, K.A., *QUIN: Integration of Inferential Operators within a Relational Database*, Department of Computer Science, University of Illinois, 1983.

[5] Winston, P.H., *Artificial Intelligence, Second Edition*, Addison-Wesley, 1983.

APPENDIX - SUNWINDOW INTERFACE

Below is a Pascal interface to the Sunwindow screen package which was used in implementing the network editor. At present the package allows a process to open a single window on the screen, and perform i/o with that window only.

Because signal processing is frequent during window operation, input is done differently to avoid race conditions and other errors associated with asynchronous signal processing. The basic idea is that all input is treated like signals via the select call, which enables the interface to pick up all the signals so that the programmer doesn't have to worry about them. This means however that input comes to the program as signals would, the program has no choice but to accept it. It's as if input is saying, "Don't call us, we'll call you." As a result, programs that use the interface must contain a procedure as described below which is called whenever a window receives input. This notification process must be enabled initially by calling the winselect procedure and can be disabled by calling the winselectdone procedure both of which are described later under PROCEDURES.

The notification procedure that the programmer is required to write should be declared as

```
procedure winprocessinput ( <paramname> : ptrinputevent );
```

The procedure *must* have this name. After the initial winselect call, winprocessinput will be called will a pointer to an inputevent record (see TYPES below) after a winselect call whenever desired input is received by the window. Types of input desired are indicated to the window system through the winnotify procedure (below).

Typically, the winprocessinput procedure will be a large case statement, which will call other procedures based on different inputs. If a program needs to read character strings from the window, this procedure must be aware of this fact (through a global state variable for example). In this state, the procedure must place characters as they are received into a global string variable until the string terminator character is read in. At this point, another global variable must be checked to determine the procedure to be called that will operate on the string. This variable must be set prior to the first character being read in.

To clarify string processing, suppose that a program has a menu with an option of "Enter Name" which allows the user to enter his name (a character string) from the keyboard. Assume that the winprocessinput procedure has recognized that this menu option has been selected. The following are the steps the program should take.

1) The winprocessinput procedure calls a getname procedure which sets up global variables to read in a string and to call a setname procedure once the string has been read in. A simple way to do this in general is by having a global input mode variable. Its initial value is none, meaning do not read character strings. When a string must be read in, winprocessinput calls a getXXXX procedure which, besides putting up a prompt and turning off mouse input, sets the input mode variable to callXXXX.
Thus the getname procedure sets the global input mode varibale to callname, then returns to winprocessinput which in turn, returns to the interface.

2) The next time the winprocessinput procedure is called, it must check to see if input mode is none. When the mode is not none, but callname for example, it will put the character from the input record into a global string variable, and increment the global index for that string so that the next character can be put in the right place. Only one global string and one global index are required by a program since only one string can be read in at a time.

3) The winprocessinput procedure continues as in 2) until the string terminator character is read in. At this point, another case statement is needed to check the input mode variable and call the appropriate procedure. In this example, a setname procedure would be called to copy the global string variable into the name variable, reset the input mode variable to none, and return to the winprocessinput procedure, which returns to the interface for continued input processing. Either this type of procedure or the getXXXX procedures should reset the global string index to zero and, if desired, blank out the global string.

4) To discontinue input processing, the winprocessinput procedure (or a procedure called from winprocessinput) should call the winselectdone procedure. When winprocessinput returns to the interface, program control will be transferred to the code immediately following the most recent winselect call.

PROCEDURES

```
procedure wininit(id, x, y : integer; var rows, cols : integer;
                  font : winfonttype;
                  var success : boolean);
        external;
(*
        attempts to run process in a window with origin at x,y
        with size rows x cols and default font.  Actual size of window
        with respect to the size of characters in font is returned
        in rows and cols.  Currently the process is assumed to be
        running inside a window already, and id, x and y are all ignored.
        additional wininit calls before a windone will have no effect.

        false is returned in success if no windows are  available.
*)

procedure winclear;
        external;
(*
        clears the entire window of its contents
*)

procedure winfont(font : winfonttype);
        external;
(*
        changes the window font to that denoted by the parameter font.
        currently winfonttype = (small, medium, mediumbold, large)
*)

(* for the following three output procedures op is a scalar with values:
        normal : the item is drawn normally (bl. on white);
        clear :  the item is erased (drawn white);
        highlight : the item is drawn inversely (wh. on bl.);
        black : the item is drawn black;
*)

procedure winchar(ch : char; row, col : integer; op : optype);
        external;
(*
        places the character ch at row, col of the window
        according to op.
*)
```

```
procedure wintext(var s : strtype; row, col : integer; op : optype);
        external;
(*
        places the string s beginning at row, col of the window
        according to op.
*)

procedure winvector(r1,c1,r2,c2 : integer; op : optype);
        external;
(*
        draws a line from r1,c1 to r2,c2 according to op
*)

procedure wininverse(x,y,width,height : integer);
        external;
(*
        inverts the region beginning at x,y extending width pixels right
        and height pixels down
*)

function winyfromrow(row : integer) : integer;
        external;
(*
        given a row corresponding to a row of text in the current font
        winyfromrow returns the starting y coordinate for text
*)

function winxfromcol(col : integer) : integer;
        external;
(*
        given a column corresponding to a column of text in the current font
        winxfromcol returns the starting x coordinate for text
*)

function wincharheight : integer;  external;
(*  returns the height in pixels of text in the current font *)

function wincharwidth : integer;  external;
(*  returns the width in pixels of text in the current font *)
```

```
(*
        The following three procedures handle menu processing.

        The first procedure, winallocmenu, allocates the necessary
        memory to the interface for a menu.  It also sets the
        title of the menu.  winallocmenu can only be called once
        for each menu.

        The second and third procedures can be called as often as
        desired.  The second procedure, winsetupmenu, determines
        the number and order of items in a menu already allocated
        by winallocmenu.

        The third procedure, winmenudisplay is a function which
        displays a menu and returns the item number of the selected
        item.
*)

procedure winallocmenu(menuid : winmenuid;
                       var title : winmenuitem;
                       maxitems : winnumitems
);
        external;
(*
        allocates a menu to be referred to in future calls as menuid.
        menu will have title as its header.
        the number of items displayed in the menu can never exceed maxitems.
*)

procedure winsetupmenu(
        menuid : winmenuid;
        numitems : winnumitems;
        var menu : winmenu
);
        external;
(*
        sets up selectable items for the menu allocated as menuid.
        numitems is the actual number of items in the menu and
        menu is an array of the items.  the first numitems in menu
        will be the items displayed upon a call to winmenudisplay (below).
*)
```

```
function winmenudisplay(menuid : winmenuid; button : buttontype) : integer;
        external;
(*
        the menu referred to by menuid will be displayed.
        button is the button which invoked the menu.
        menudisplay waits for this button to be released as
        indication that an item has been selected.
        thus this function may be called other than in
        response to a button press.  winmenudisplay returns
        an index to the array which was used to setup the menu
        which indicates the item selected by the user. If no item
        is selected, 0 is returned.
*)

procedure winnotify(ascii, button, move: boolean);
        external;
(*
        each parameter set to true will cause the winprocessinput procedure
        to be called for the corresponding input.
        each parameter set to false will prevent calls to winprocessinput
        on the corresponding input.
        default settings are keyboard = true, button = false, move = false.
*)

procedure winselect;
        external;
(*
        call this procedure to begin input processing.
        a call to winselect is best thought of as entering a
        loop which waits for input which, when received, causes the
        winprocessinput procedure to be called.
        calling winselectdone (below) causes the loop to exit.
        additional winselect calls after a winselect will be ignored.
*)

procedure winselectdone;
        external;
(*
        Causes a return from winselect (see above).
*)

procedure windone;
        external;
(*
        graceful termination by clearing screen and deallocating
        associated storage.
*)
```

```
function winconfirm(var s : strtype) : boolean;
        external;
(*
        the string s is displayed to the user as a prompt
        s should be something the user can confirm
        actions for confirming s are displayed by the function.
        if the user performs the confirming action (left-button press)
        the function returns true, otherwise false is returned
*)

procedure winmessage(var s : strtype);
        external;
(*
        similar to winconfirm except that no confirmation is acknowledged.
        winmessage displays s and waits for the user to do anything.
        s should include a something like 'press any key to continue'
*)

function strlen(var s : strtype) : integer;
        external;
(*
        for the builtin C routine strlen.
        returns the number of characters in s
*)
```

TYPES

```
const
  winmenuitemlen = 32;
  winmenusize = 32;
  winmaxmenus = 32;

type
  strtype = packed array[1..100] of char;
  buttontype = (left,middle,right);
  postype = record
    row,col : integer;
    end;
  winiotype = (keyboard, mousebutton, mousemove);
  winfonttype = (small,medium,mediumbold,large);
  ptrinputevent = ^inputevent;
  inputevent = record
    pos : postype;
    case inputtype : winiotype of
      keyboard : (ch : char);
      mousebutton : (button : buttontype);
    end; (* inputevent *)
  optype = (normal, clear, inverse, black);
  winmenuid = 1..winmaxmenus;
  winmenuitem = packed array[1..winmenuitemlen] of char;
  winnumitems = 1..winmenusize;
  winmenu = array[winnumitems] of winmenuitem;
```

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-F-85-934	2.	3. Recipient's Accession No.

4. Title and Subtitle

Editing Networked-Structured Knowledge Bases in the ADVISE System

5. Report Date: February 1985

6.

7. Author(s)

Thomas D. Channic

8. Performing Organization Rept. No.

9. Performing Organization Name and Address

Department of Computer Science
University of Illinois
Urbana, IL

10. Project/Task/Work Unit No.

11. Contract/Grant No.

DCR 84-06801
N00014-82-K-0186

12. Sponsoring Organization Name and Address

National Science Foundation
Washington, D.C.

Office of Naval Research
Washington, D.C.

13. Type of Report & Period Covered

14.

15. Supplementary Notes

16. Abstracts

Until recently, knowledge bases in the ADVISE meta-expert system have had to be edited by text editing or by changing the program that creates the knowledge base. This paper discusses a network editor for ADVISE knowledge bases which can edit a knowledge base directly. The network editor has been implemented on a Sun Microsystems Workstation, and is screen-oriented and menu-driven. The latter part of this report serves as a user's guide for the editor.

17. Key Words and Document Analysis. 17a. Descriptors

knowledge base editing
expert system interface

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages 52
	20. Security Class (This Page) UNCLASSIFIED	22. Price