USING THE EXPERT'S DIAGRAMS AS A SPECIFICATION OF EXPERTISE

Technical Report AIP - 50

Stephen Casner & Jeffrey Bonar

Learning Research & Development Center and
Intelligent Systems Program
University of Pittsburgh
Pittsburgh, Pa. 15260

1 July 1988

This research was supported by the Computer Sciences Division, Office of Naval Research and DARPA under Contract Number N00014-86-K-0678. Reproduction in whole or in part is permitted for purposes of the United States Government. Approved for public release; distribution unlimited.

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; Distribution unlimited
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AIP - 50	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Carnegie-Mellon University 6b. OFFICE SYMBOL (*If applicable*)	7a. NAME OF MONITORING ORGANIZATION Computer Sciences Division Office of Naval Research
6c. ADDRESS (*City, State, and ZIP Code*) Department of Psychology Pittsburgh, Pennsylvania 15213	7b. ADDRESS (*City, State, and ZIP Code*) 800 N. Quincy Street Arlington, Virginia 22217-5000
8a. NAME OF FUNDING / SPONSORING ORGANIZATION Same as Monitoring Organization 8b. OFFICE SYMBOL (*If applicable*)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER N00014-86-K-0678
8c. ADDRESS (*City, State, and ZIP Code*)	10 SOURCE OF FUNDING NUMBERS p4000ub201/7-4-86 PROGRAM ELEMENT NO. N/A; PROJECT NO. N/A; TASK NO. N/A; WORK UNIT ACCESSION NO N/A
11. TITLE (*Include Security Classification*) Using the Expert's Diagrams as a Specification of Expertise (Unclassified)	
12. PERSONAL AUTHOR(S) Casner, Stephen and Bonar, Jeffrey	
13a. TYPE OF REPORT Technical	13b. TIME COVERED FROM 86Sept15 TO 91Sept14
14. DATE OF REPORT (*Year, Month, Day*) 1988 July 1	15. PAGE COUNT 24
16. SUPPLEMENTARY NOTATION Proceedings of the 1988 Workshop on Visual Languages, October 10-13, Pittsburgh, PA	

17 COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (*Continue on reverse if necessary and identify by block number*)
Visual languages, Diagrams

19. ABSTRACT (*Continue on reverse if necessary and identify by block number*)

See reverse side

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION
22a NAME OF RESPONSIBLE INDIVIDUAL Dr. Alan L. Meyrowitz	22b TELEPHONE (*Include Area Code*) (202) 696-4302 22c. OFFICE SYMBOL N00014

This work explores the use of diagrams in generating executable specifications of expert knowledge. We make the observation that experts frequently use diagrams as an efficient means of communicating detailed information. For some types of information diagrams might offer the expert an alternative to the high cost of understanding existing knowledge representation formalisms. We are interested in accomplishing three things: 1) understanding the diagramming techniques used by domain experts to encode detailed information in a restricted type of diagram called a relational diagram; 2) characterizing a set of notions that experts frequently encode in relational diagrams; 3) developing an environment that allows experts to partially construct a formal specification of problem domain knowledge by drawing relational diagrams. We describe BOS, a diagramming tool that allows domain experts to build a customized set of diagramming conventions suitable to their problem domain. Diagrams drawn with BOS generate formal specifications that reduce the need to establish the diagram's meaning through accompanying text or verbal explanation. BOS is currently able to generate frames and rules from an interesting set of relational diagrams that allow the use of spatial arrangement and connectivity to represent notions about problem domain entities, part of relations, constraints, temporal ordering, and procedural steps.

Abstract

This work explores the use of diagrams in generating executable specifications of expert knowledge. We make the observation that experts frequently use diagrams as an efficient means of communicating detailed information. For some types of information diagrams might offer the expert an alternative to the high cost of understanding existing knowledge representation formalisms. We are interested in accomplishing three things: 1) understanding the diagramming techniques used by domain experts to encode detailed information in a restricted type of diagram called a relational diagram; 2) characterizing a set of notions that experts frequently encode in relational diagrams; 3) developing an environment that allows experts to partially construct a formal specification of problem domain knowledge by drawing relational diagrams. We describe BOS, a diagramming tool that allows domain experts to build a customized set of diagramming conventions suitable to their problem domain. Diagrams drawn with BOS generate formal specifications that reduce the need to establish the diagram's meaning through accompanying text or verbal explanation. BOS is currently able to generate frames and rules from an interesting set of relational diagrams that allow the use of spatial arrangement and connectivity to represent notions about problem domain entities, part of relations, constraints, temporal ordering, and procedural steps.

Introduction

This work is concerned with the use of diagrams in generating specifications of expert knowledge such as those used by expert reasoning systems. It is well understood that learning to use existing knowledge representation languages is difficult [Anderson, 1984; Zhang, 1987]. Previous work on diagrams characterizes several psychological properties of diagrams that make them an appealing medium of expression [Larkin and Simon, 1987; Fitter and Green, 1979]. We observe that in many problem domains experts frequently use diagrams as an efficient means of communicating detailed information. We would like to show that for some types of information, diagrams offer an alternative for expressing problem domain knowledge, and that experts already have an established set of skills and conventions for using diagrams.

The present work is restricted to a particular type of diagram called a *relational diagram*. We define this notion and discuss two ways of graphically encoding information in relational diagrams. We present a series of textbook examples illustrating how diagramming techniques are associated with problem domain concepts to encode information in relational diagrams. We describe BOS, a diagramming tool that allows users to build simple executable specifications of problem domain knowledge by implementing and using a set of customized diagramming conventions suited to their problem domain.

Relational Diagrams

Hegarty and Just (1988) distinguish two types of diagrams that are commonly used to accompany text or verbal explanations. *Realistic diagrams* depict the subject matter similar to its appearance in real life and preserve many of the details of its appearance. *Relational diagrams* abstractly represent entities in a domain, their properties, and relationships between them. A relational diagram may contain a set of objects and links between them. Graphical objects are used to represent entities in the domain. Links can be drawn between objects to specify relationships between objects. Relational diagrams differ from simple graphs in that connectivity is not the only way to express relationships between entities. For instance, a spatial arrangement of entities in a diagram can be used to encode information about their relative physical positions, or importance relationships between them.

Previous Work

Harel (1987) and Mackinlay (1986) describe visual languages that capture some of these relational notions. Harel's visual language allows interface designers to build functional specifications of complex systems by using a customized set of diagramming conventions. These conventions are defined by creating visual ways of expressing each of a set of constructs in a non-diagrammatic formalism. A limitation of Harel's system is that using the visual language still requires an understanding of the complex formalism since it is simply an alternative (visual) syntax for the same set of complex notions.

Mackinlay's intelligent presentation tool defines a set of graphical languages that can be assigned a variety of interpretations when designing visual presentations of relational information. The set of notions described in

Mackinlay's work is general and can be flexibly used to encode many relational notions. However, for our purposes it is not clear that domain practitioners would be able to understand how these elegant notions must be combined to arrive at the domain-specific concepts they wish to express.

A basic point of both works is that visual presentations can be defined as sentences in a formal visual language that has a precise syntax and semantics. Our goal is to enable the domain expert to define their own visual language that allows them to use familiar diagramming conventions to build specifications of knowledge by drawing relational diagrams.

Encoding information in relational diagrams

This part of the work focuses on identifying a set of available techniques for graphically encoding information in the human-computer interface (a diagram syntax), and characterizing a set of useful notions the domain expert wishes to express (a diagram semantics). Bertin (1983) describes a set of techniques that includes spatial arrangement, connectivity, color, shape, texture, size, and animation. The work described here focuses on the use of spatial arrangement and connnectivity. The series of examples that follows is intended to convey two points: 1) spatial arrangement and connectivity can be used to encode a variety of problem domain notions in relational diagrams; and 2) an understanding of these conventions can be used to guide the design of a diagramming tool that allows domain experts to implement familiar sets of diagramming conventions and use them to build formal specifications of domain knowledge through diagramming.

Spatial Arrangement

Diagrams frequently relate the spatial arrangement of objects to the meaning of the diagram. For example, spatial arrangement can be used to represent the notion of physical location in the real world. This representation also encodes information about distance and orientation that can be quickly retrieved from the diagram.

Spatial arrangement is also used to describe properties or attributes of entities.

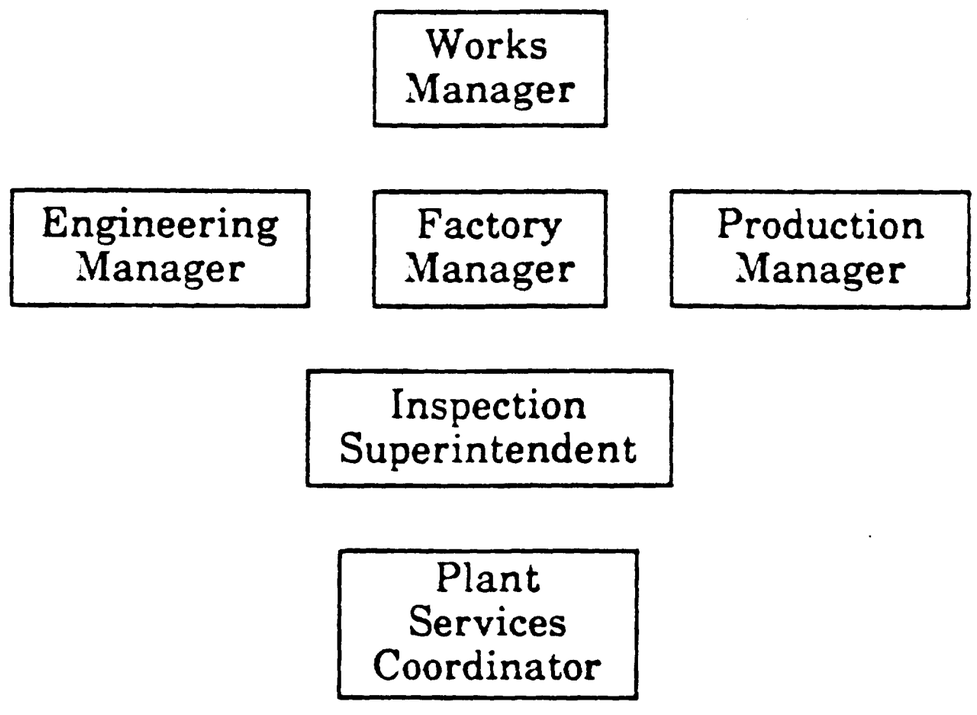


Figure 1: Spatial Arrangement

In Figure 1, the author ties the notion of "rank" to the use of the spatial arrangement. For instance, the placement of the Inspection Superintendent above the Plant Services Coordinator indicates that he/she has higher rank.

Spatial arrangement might also be used to convey notions of temporal order. For example, placing one "event" object to the right of another might indicate that the first is to occur before the second. Football diagrams use spatial arrangement to represent information about situations, procedures, and strategy.

An understanding of the use of spatial arrangement suggests a strategy for its implementation. Mackinlay (1986) characterizes two primitive graphical languages, *HorzPos and VertPos*, based on the location of an object along the horizontal and vertical axes. We can assign an interpretation to the diagram given in Figure 1 by imposing a vertical axis *(VertPos)* on the diagram. Using the axis we can associate a scalar value with some attribute of every entity represented in the diagram by simply determining its position along the vertical axis. In Figure 1 the scalar value represents that each manager's rank.

An important problem must be solved before we can implement these conventions. There must be some way of allowing the expert to indicate whether or not the spatial arrangement convention is being used, and what it is being used to signify. In the diagramming tool described below, the meanings assigned to the use of spatial arrangement is collaboratively defined by the domain expert and programmer to match a diagramming convention familiar to the domain expert.

Connectivity
(links)

Relational diagrams use links to indicate that two or more entities exist in some relationship. In Figure 2, links are used to represent constraints on generating an acceptable employee work schedule for a retail store. That is, links depict the allowable relationships between employees, jobs, and time slots.

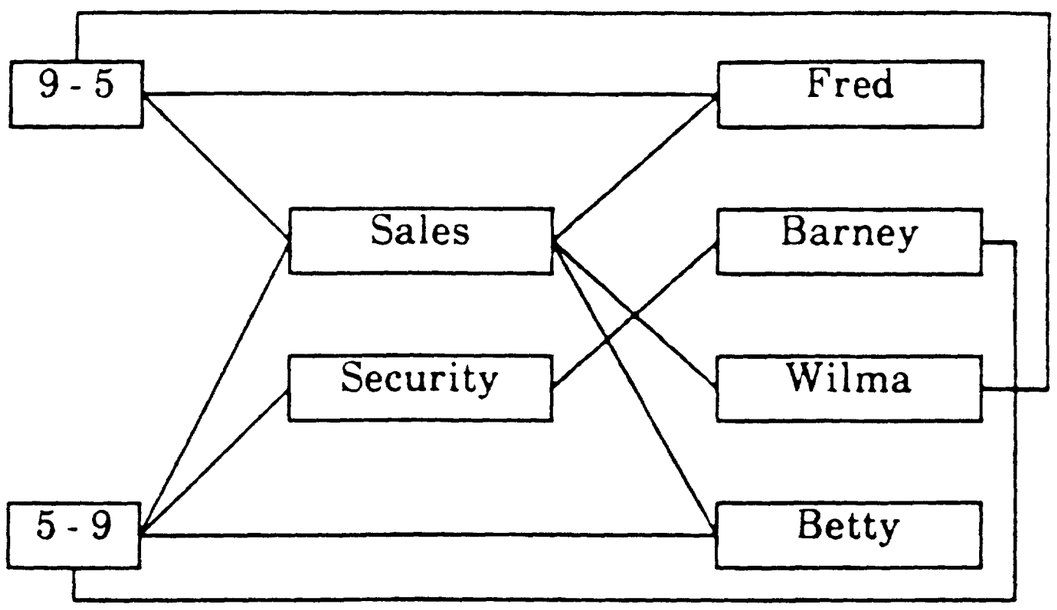


Figure 2: Constraint Diagram

Figure 2 uses links to indicate the constraint relationships. The meaning of links are pre-established in an accompanying text, through verbal explanation, or through a shared understanding between those who use the diagram. A problem with implementing links in a formal language is that we need a way of indicating how the links are to be interpreted.

Gantt charts are a second example of diagrams that use links. Gantt charts are used in the business planning domain to represent sequences of decisions and events. The following Gantt chart depicts a simple marketing plan for AI products:

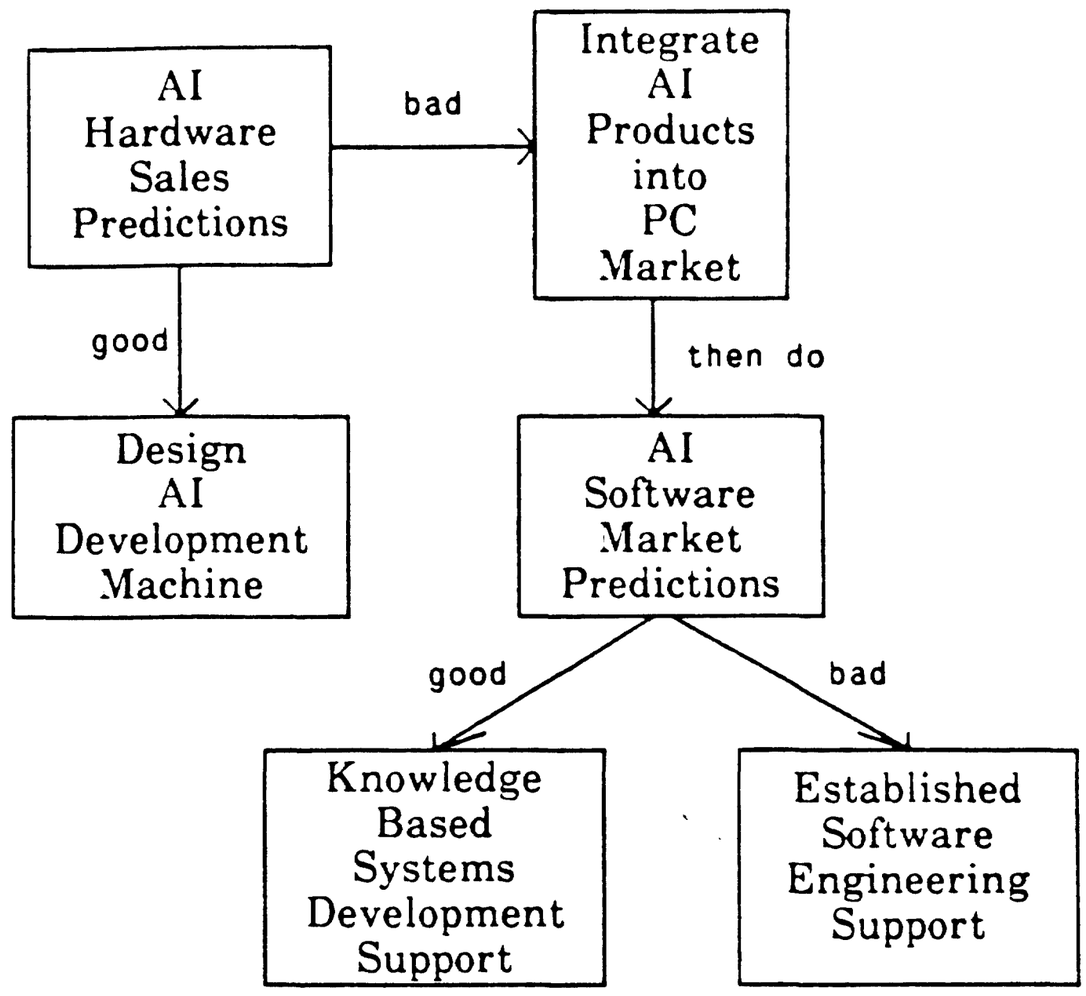


Figure 3: Procedural Steps Diagram

The links used in Figure 3 carry names that mnemonically suggests their meaning. Of course, this convention relies on the individual's understanding of the mnemonic name and we still require a way of allowing the user to attach a formal interpretation to them.

BOS: Toward a relational diagram language for specifying expert knowledge

We have built a diagramming tool called BOS (pronounced *"boz"*) to apply the analyses from the previous section to the design of a diagramming tool that enables the expert to define customized diagramming conventions and use them to build specifications of problem domain knowledge.

BOS's Capabilities

BOS allows the user to define customized diagramming conventions by assigning interpretations to the use of spatial arrangement and links. To suit the problem domain described in the example below, we have implemented the following interpretations: temporal order, part of relationships, constraints, and procedural steps. Each of the interpretations has a formal definition associated with it. BOS allows the expert to draw simple relational diagrams using the defined conventions from which it then builds a knowledge base expressed in frames and rules [Winston and Horn, 1984]. For every object and link placed in the diagram a frame template is inserted in the knowledge base. The frames are filled in with the interpretations attached to the use of spatial arrangement and links. For example, for the diagram in Figure 1, each time a new object is placed in the diagram a "rank" attribute is added to the framce corresponding to that object and the appropriate value is filled in. A knowledge base is extended as new objects and links are added to the diagram. The additions to the knowledge base are not reflected in the diagram in any way, and the expert need not necessarily be concerned with their existence nor their relationship to the diagram. The set of diagram conventions is the only specification language the expert must understand.

An example: Building a simple critic for novices' Pascal while loops

The following example illustrates BOS's current set of features. The example problem domain is debugging simple Pascal While loops. We want to use diagrams to express some of the processes involved when an instructor (an expert programmer) views and identifies bugs in a novice's Pascal loop. The example shows that diagrams can be used to build a knowledge base

consisting of a set of frames and rules that comprise a simple specification of the instructor's debugging strategies. The example was generated cooperatively with an expert Pascal programmer who has no experience with frames or rule-based knowledge representation techniques.

The example presented here is restricted to the task of identifying a few common bugs frequently found in novices' looping constructs. The example only considers simple Read/Process looping problems such as:

Read in a list of numbers until 99999 is encountered and report the sum of the numbers.

To accomplish this we would like a set of diagramming conventions that allows us to encode the following things:

1) the main *entities* in a simple read/process Pascal loop such as the primer read, the variable being read, the condition test, the variable being tested, etc.

2) the order in which the parts of the loop occur (i.e., the "condition" part comes before the "read next" part).

3) a set of *part of relationships* to indicate that certain variables or statements are associated with the main parts of the loop (i.e., the variable "count" is part of the condition statement).

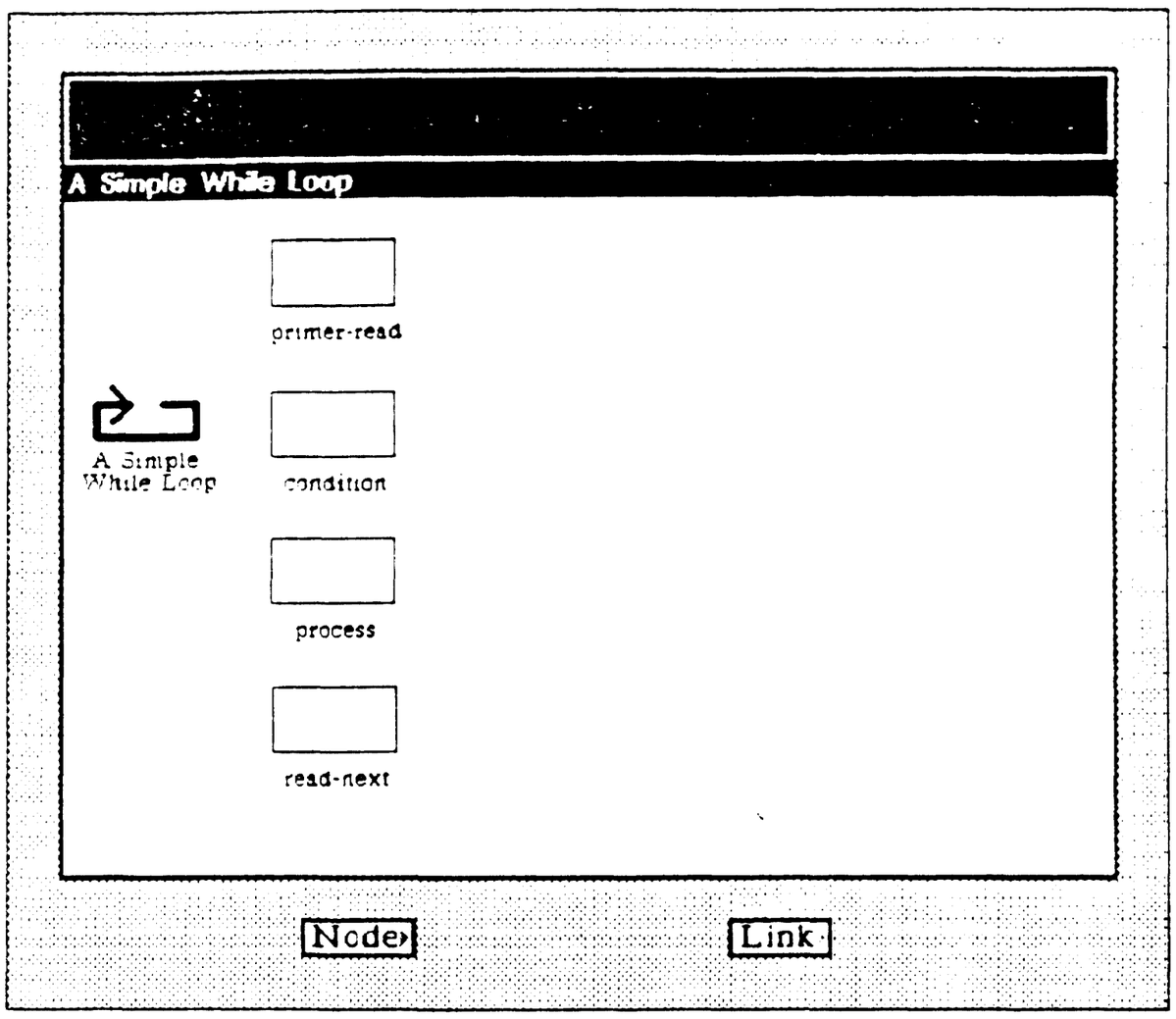
4) a set of *constraints* that hold among the parts of the loop. The diagram contains constraint links that indicate that, in order for the loop to be correct, various parts of the loop must agree with, or relate to, other parts in some specified way.

5) a set of *procedural steps* links indicating how the constraints should be applied in debugging a particular loop.

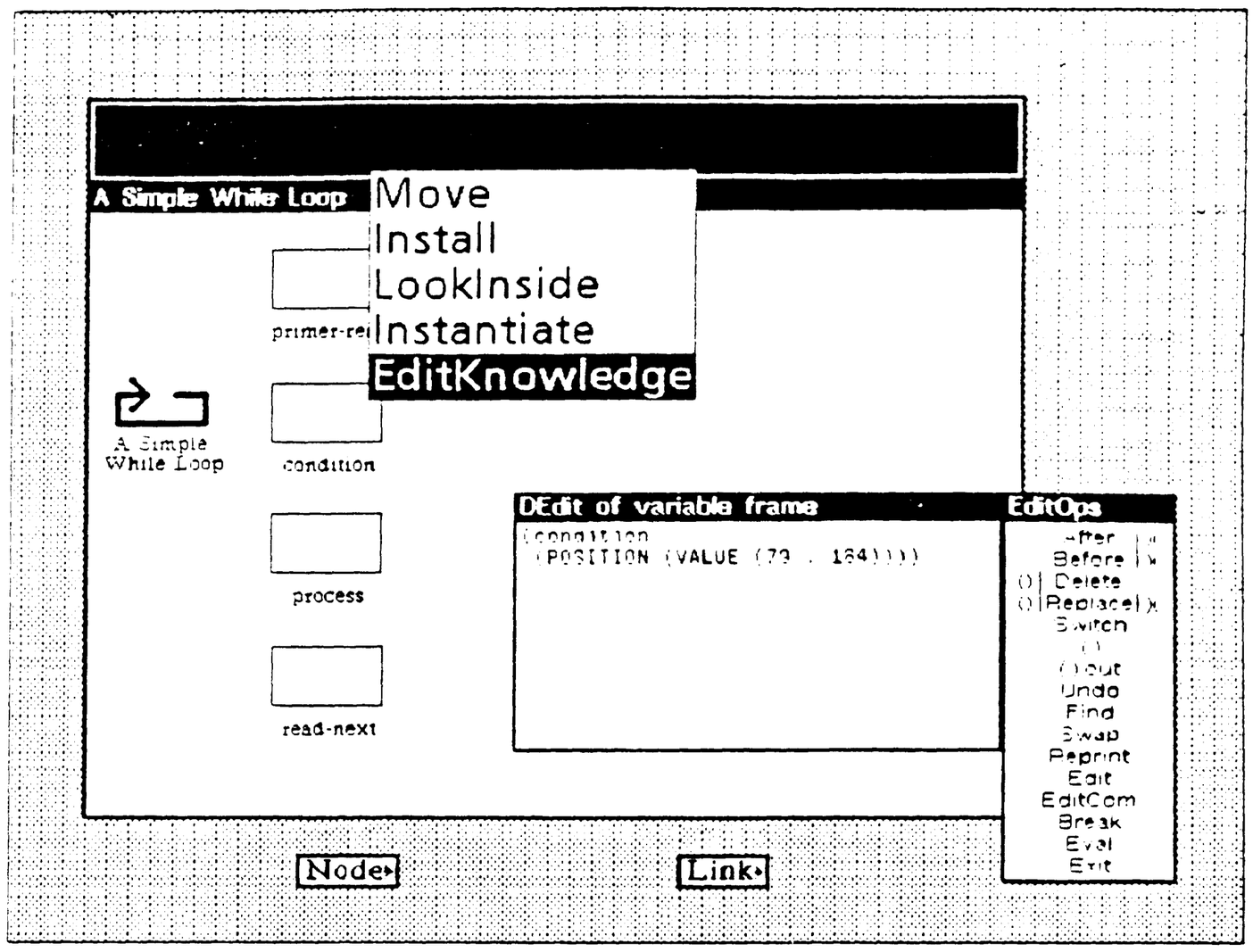
We defined a customized set of diagramming conventions as follows. The use of vertical arrangement is associated with the notion of temporal order. That is, whenever an object is placed above another object, that means the first objects occurs before the second. We defined a set of links that are associated with formal definitions of part of relations, constraints, and conditionals. For each new node and link added to the diagram, BOS inserts the associated definition in the frames in the knowledge base. The frames are shown to the reader throughout the example as the diagram and knowledge base become more complex.

Diagramming the Parts of a Typical Loop

In our session with the expert, we started by specifying a prototypical Pascal loop. The diagram below contains objects to represent some of the major components of the loop. The expert may use the default "box" object or elect to draw an icon to represent the object. The expert is also asked to provide a name for each object. In our example, we drew an icon for the "loop" node and used the default for the other objects.

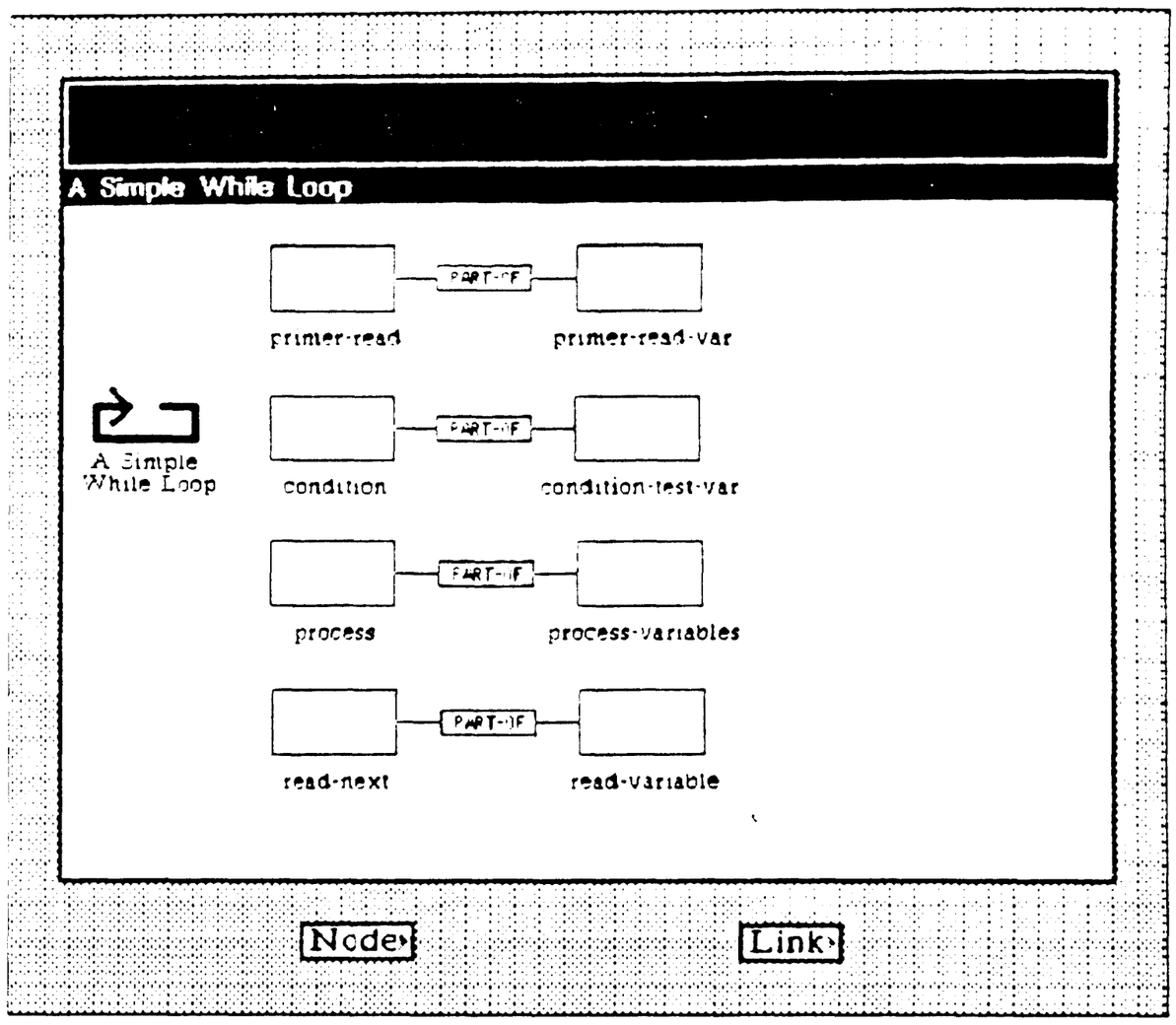


For each object in the loop diagram, BOS places a frame in the knowledge base to represent that diagram object. We can view this frame by selecting the **EditKnowledge** option from the diagram object menu. Here we view the frame associated with the `condition` part of the loop:

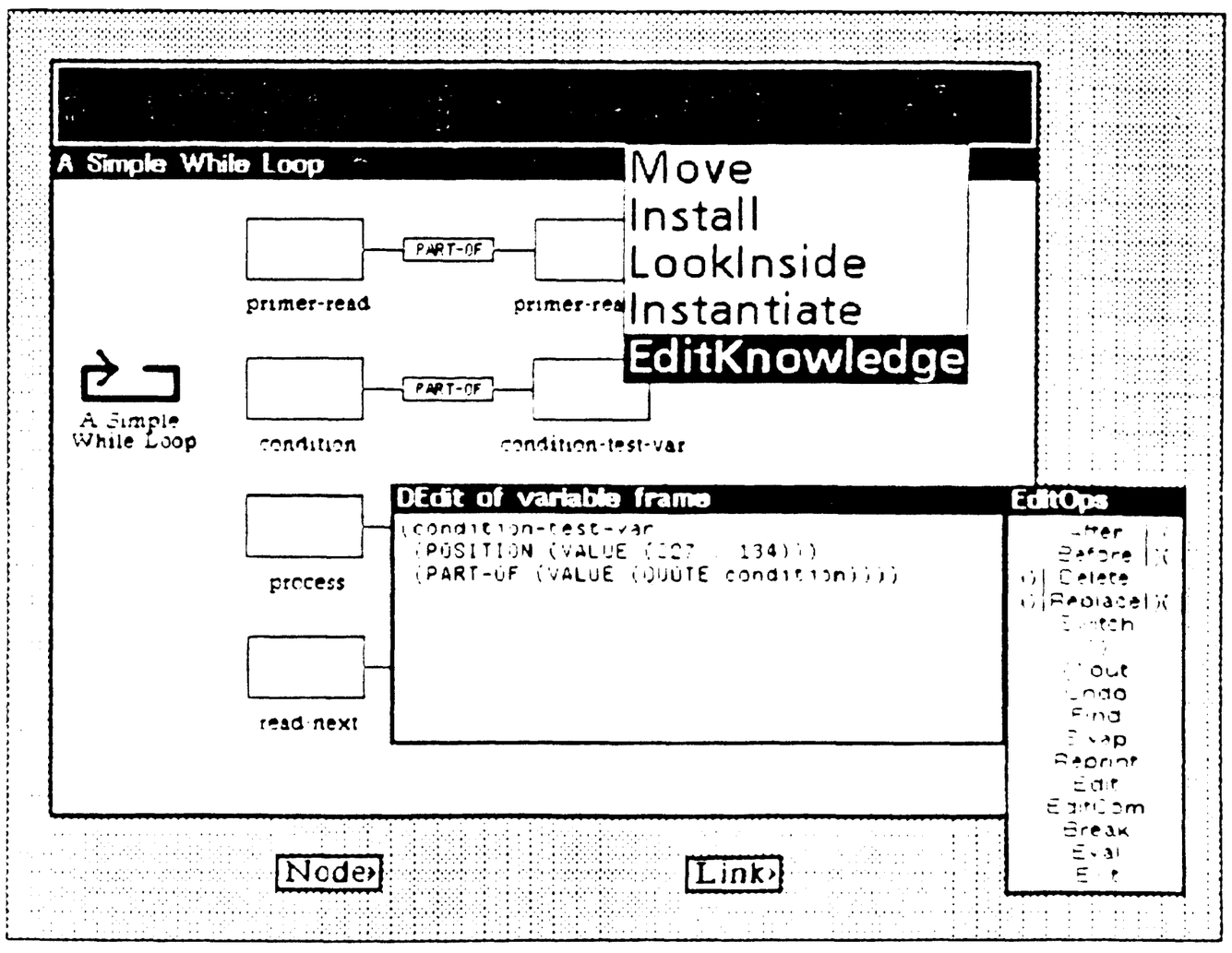


Note that the `condition` frame contains only one slot at this point. This is because we have not yet specified any other information about `condition`, such as attributes it may have, or how it is related to other parts of the loop. The `condition` frame is gradually extended as more links are included indicating how it relates to other parts of the loop. The `order` slot represents the association of the object's position in the diagram with the notion of temporal order.

A object can be designated as a part of another object by using the PART-OF link we defined for our problem domain.



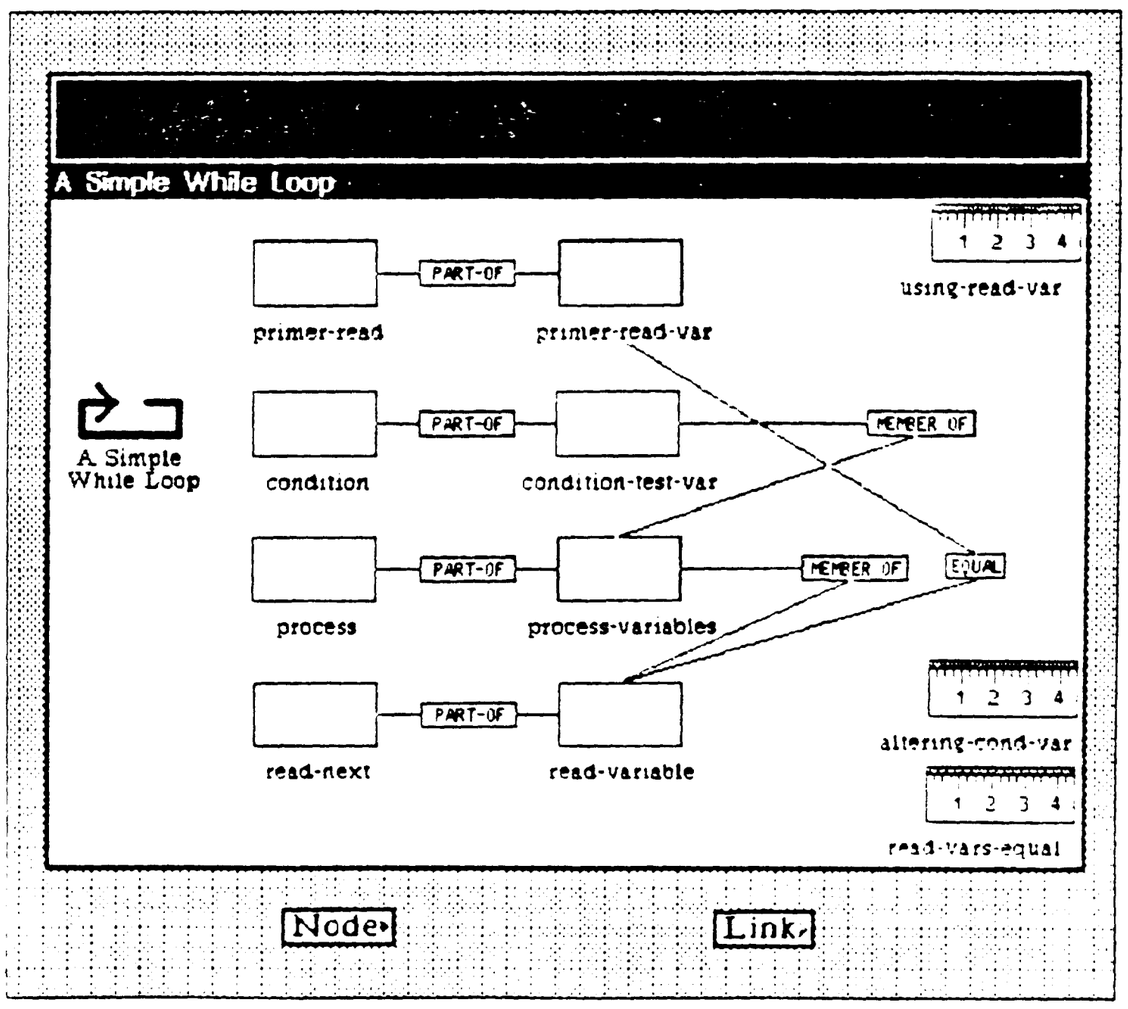
We can inspect the frame that is associated with these objects as well. Here we choose the **EditKnowledge** option for the `condition-test-var` object:



Note that adding the PART-OF link has inserted an appropriate slot in the `condition-test-var` frame.

Specifying Constraints

Constraints are drawn to indicate that some part of the loop must exist in some relationship with other parts. For example, constraints can be used to indicate that some variable must have a particular value, or be equal to another variable. Recall we defined a set of constraint links for our problem domain. The following diagram includes some simple constraints that the While loop must obey in order to be correct:



The **EQUAL** constraint link specifies that the variable read before the loop should be the same variable read inside the loop body.

The uppermost **MEMBER** link states that the variable that is tested in the condition should be modified somewhere in the body of the loop, if the loop is to terminate.

The lower **MEMBER** link specifies that the variable read in the loop body should somehow be used in the "process" part of the loop.

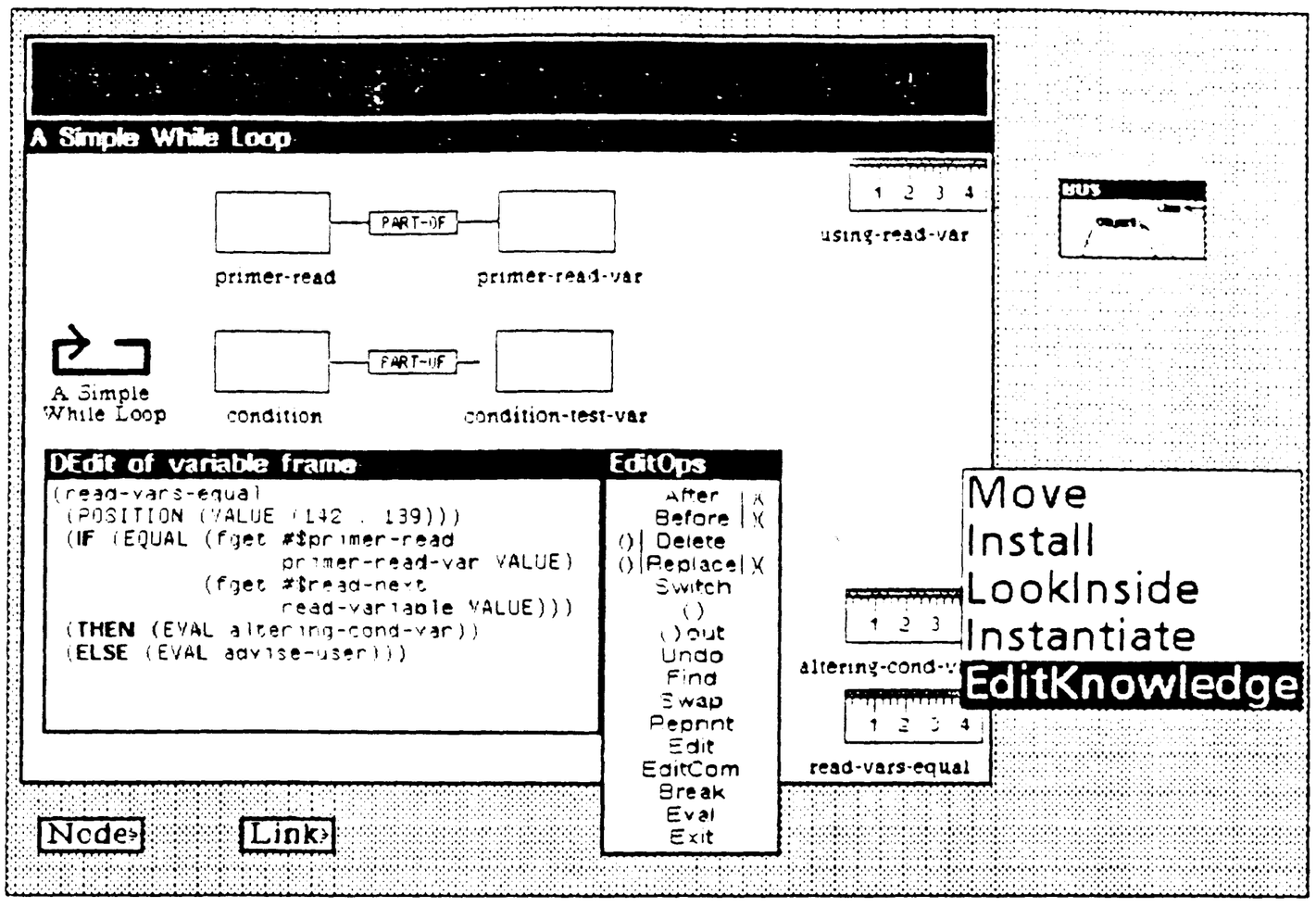
As part of our formulation of the constraint definition, for each constraint drawn in the diagram, BOS does two things. First, the frames corresponding to each of the constrained entities are extended to include this new aspect of their definition. For instance, the frame associated with the `read-variable` object now includes a slot called **EQUAL** that is assigned the value `primer-read-var`. Second, BOS makes the following inference:

> Since a constraint has been added to the diagram indicating that some entity must obey some constraint, insert a rule in the knowledge base that checks to see if this constraint is obeyed.

The knowledge base now has one piece of procedural knowledge that can be used to diagnose a buggy loop. Given an example of a While loop, the knowledge base now has a rule for checking to see that the loop obeys this constraint. Diagrammatic representations of these rules are also added to the diagram. These are the "ruler" icons in the diagram. The expert is asked to give each rule a name. The screen above contains three rules that correspond to the constraints: `using-read-var`, `altering-cond-var`, and `read-vars-equal`.

Rules are a variation of a typical production rule. Rules have three parts: IF, THEN, and ELSE. For each constraint placed in the diagram, an IF clause of a rule is filled in. The IF clause describes the condition that must be met in order for the constraint to be satisfied. **EditKnowledge** can be selected from any of the rules to

view the description that has been inserted in the knowledge base by BOS:



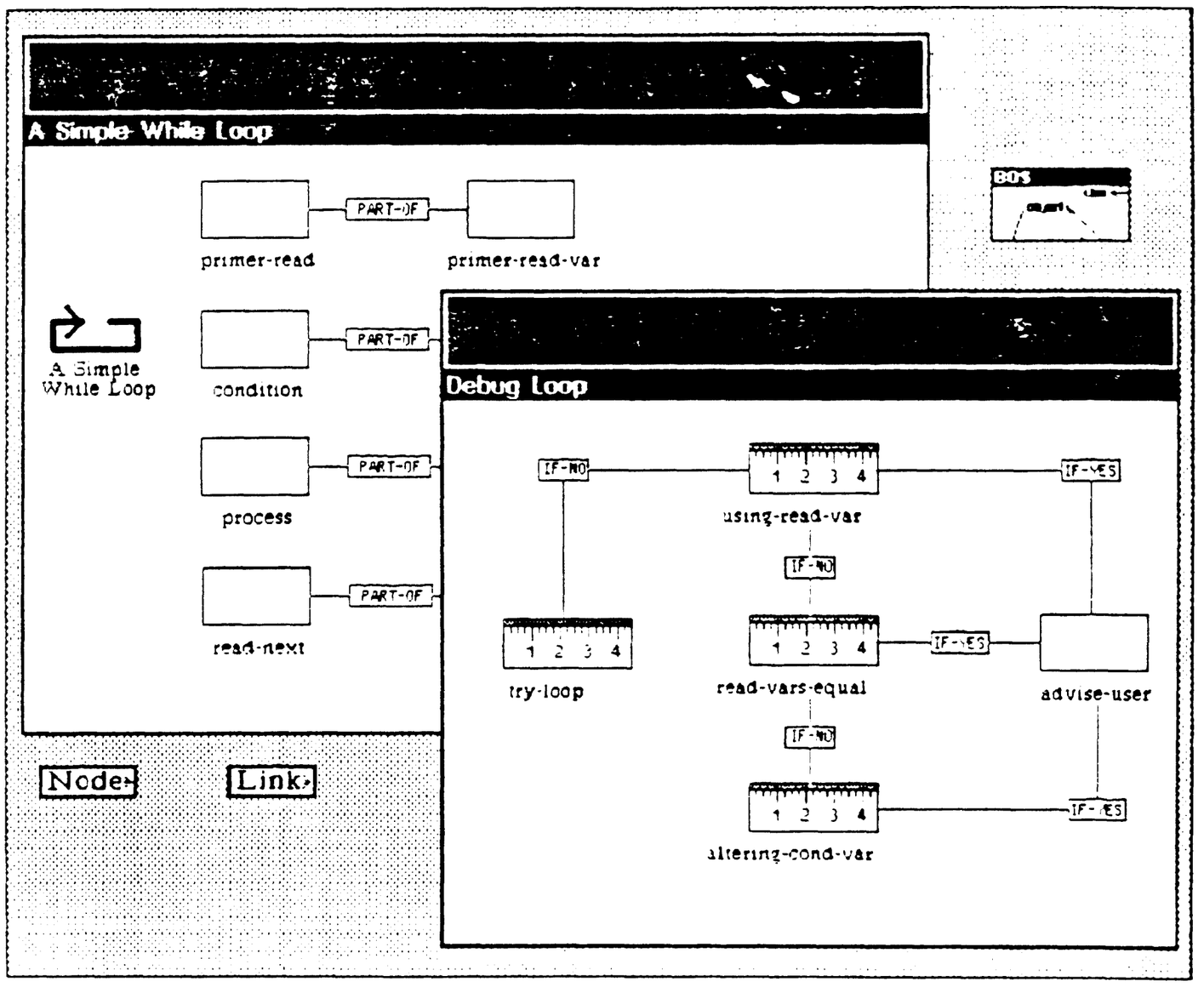
Note that only the IF part of the rule has been specified. The constraint only specifies the condition that must be met in order for the constraint to be satisfied. It says nothing about how or when the rule should be applied, in what order, or what action is to be taken if the rule fires.

Specifying Procedural Steps

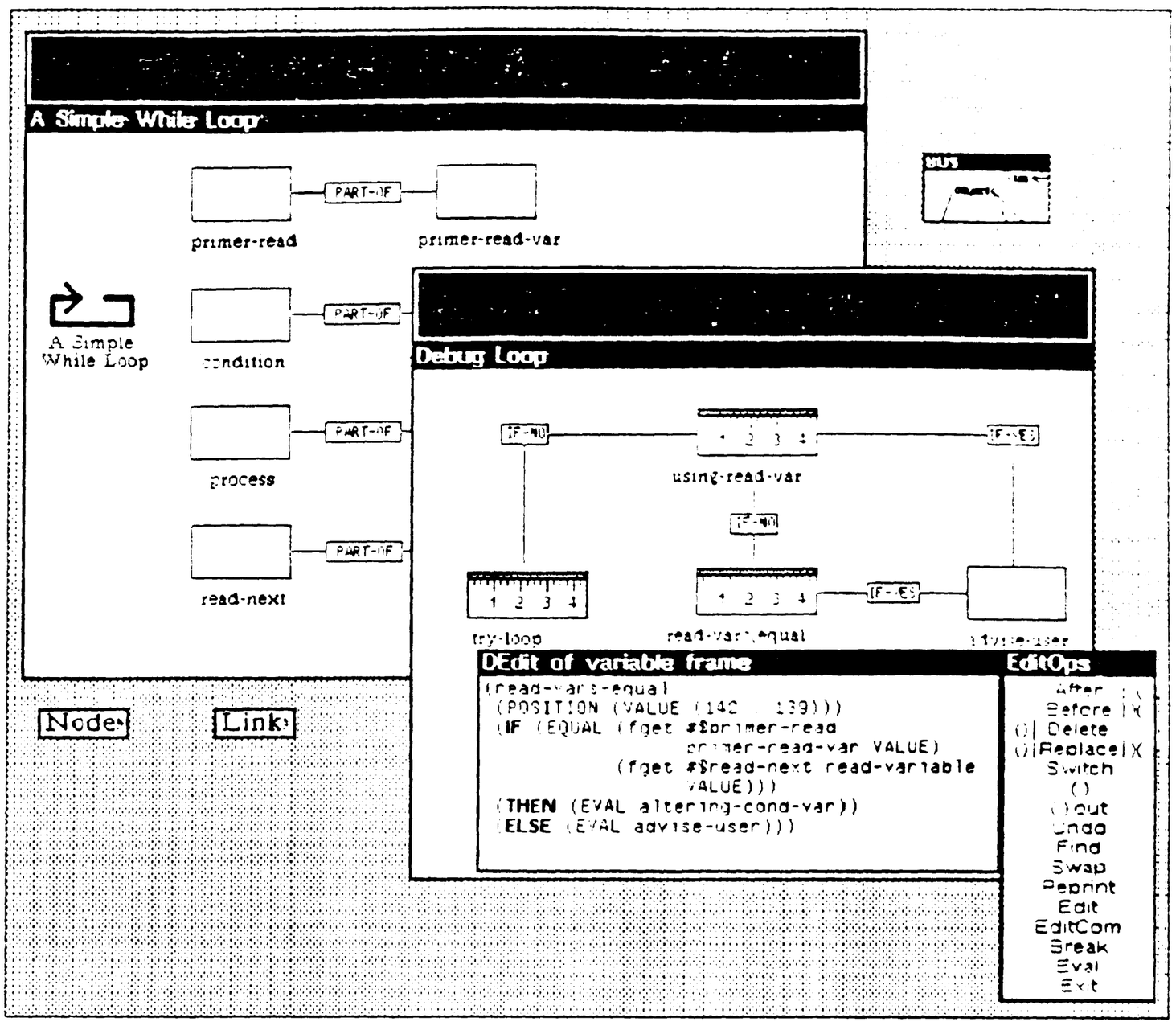
In the customized diagram language we specified for our domain, specifying procedural steps is how the THEN and ELSE parts of the rules are filled in. The partially specified rules are used to formulate a procedure that says how the rules are to be applied in debugging a Pascal loop. This is done by using the procedural step links provided by BOS. BOS currently implements three: IF-YES, IF-NO, and SEQUENCE. An IF-YES link between two rules indicates that if the first rule fires, the second rule should be tried. An IF-NO link indicates that

the second rule should be tried if the first rule fails to fire. A **SEQUENCE** link between two rule icons indicates that both rules are to be tried in sequential order.

For purposes of organization, BOS allows multiple diagrams to be created. Parts of any diagram can be moved or copied to any other diagram. In this case we create a separate diagram for specifying the procedural steps. Our procedure specification looks like:



Note that we have added an additional rule, `try-loop`, that initially runs the loop to see if it is correct (BOS allows rules to be inserted "manually" as well). We can select **EditKnowledge** on the `read-vars-equal` rule and see that the THEN and ELSE clauses of the rule have been filled in to reflect the specification of the procedural sequence:



This rule states:

IF the variables are the same
THEN try another rule (this isn't the problem)
ELSE advise user that the variables must be the same

The Knowledge Base

The Appendix shows the entire knowledge base that BOS produces from the diagram. The code generated follows the frame implementation found in Winston and Horn (1984). In order to run the program, it remains to encode an example loop in list notation, indicate that `try-loop` is to be applied to it, and fill in the messages that are to be reported to the user when bugs are found (`advise-user`).

Discussion

Our example suggests that, for some types of problem domain knowledge, diagrams offer a plausible alternative for building executable specifications of expert knowledge. There appear to be two obstacles to processing a wider variety of relational diagrams. First, we need to better understand a more complete set of graphical techniques for encoding information in diagrams in the human-computer interface. Second, a better understanding of the types of notions that domain experts express with diagrams would benefit from a detailed study of the use of diagrams in diagram-intensive domains. The first author is presently undertaking a study of the use of diagramming in VLSI design and football.

Diagrammatically representing procedural steps in the manner described above becomes problematic when procedures become larger. Some procedures might involve hundreds or even thousands of rules. Clearly, it would be impractical to attempt to place all rules and links in one diagram, hence, we must develop an alternative strategy for managing this complexity.

In the context of knowledge engineering, it appears that diagrams may provide a medium of expression that enables the expert to take a more participative role in designing the knowledge base without having to understanding all of the details of the knowledge representation scheme being used. We are also exploring other possible benefits of using diagrams such as a means of improving the expert's recall for domain concepts and relations.

Acknowledgements

This research was sponsored by the Office of Naval Research, University Research Initiative, Contract No. N00014-86-K-0678. We are grateful to Stellan Ohlsson, William Oliver, and Leila Wallace for helpful ideas about relational diagrams. Clayton Lewis' early ideas about "representations" of expert knowledge inspire me to pursue diagrams. Robert Cunningham's programming expertise enabled some of the finer points of BOS's interface.

References

Anderson, John R., Farrell, R., and R. Sauers, "Learning to program in LISP," *Cognitive Science* 8, 87-129.

Bertin, Jacques, *Semiology of Graphics*, W. J. Berg, Transl., University of Wisconsin Press, Milwaukee, WI, 1983.

Fitter, M. and T. R. G. Green, "When do diagrams make good computer languages," International Journal of Man-Machine Studies (1979) 11, 235-261.

Harel, David, "On visual formalisms," Carnegie-Mellon Technical Report CMU-CS-87-126, 1987.

Hegarty, Mary, and Marcel Just, "Understanding machines from text and diagrams," to appear in H. Mandl and J. Levin (Eds.) *Knowledge Acquisition from Text and Picture*, Amsterdam: North Holland.

Larkin, J, and Simon, H, "Why a diagram is sometimes worth 10,000 words," *Cognitive Science* 11, 1987, pp. 65-99.

Mackinlay, Jock, "Automating the design of graphical presentations of relational information," *ACM Transactions on Graphics* **5** (2), April 1986, 110-141.

Winston, P.H., and B. Horn, *Lisp*, Reading, MA: Addison-Wesley, 1984.

Zhang, Guo-Jun, "Learning to program in OPS5," unpublished PhD Thesis, Carnegie-Mellon University, 1987.

APPENDIX: KNOWLEDGE BASE GENERATED BY DIAGRAM

```
(primer-read
   (POSITION (VALUE (79 . 258)))
   (PART (VALUE (QUOTE primer-read-var))))

(condition
   (POSITION (VALUE (79 . 184)))
   (PART (VALUE (QUOTE condition-test-var))))

(process
   (POSITION (VALUE (79 . 112)))
   (PART (VALUE (QUOTE process-variables))))

(read-next
   (POSITION (VALUE (79 . 39)))
   (PART (VALUE (QUOTE read-variable))))

(primer-read-var
   (POSITION (VALUE (228 . 258)))
   (PART-OF (VALUE (QUOTE primer-read)))
   (EQUAL (VALUE (QUOTE read-variable))))

(condition-test-var
   (POSITION (VALUE (228 . 184)))
   (PART-OF (VALUE (QUOTE condition)))
   (MEMBER (VALUE (QUOTE process-variables))))

(process-variables
   (POSITION (VALUE (228 . 112)))
   (PART-OF (VALUE (QUOTE process)))
   (CONTAINS (VALUE (QUOTE condition-test-var)))
   (CONTAINS (VALUE (QUOTE read-variable))))

(read-variable
   (POSITION (VALUE (228 . 39)))
   (PART-OF (VALUE (QUOTE read-next)))
   (EQUAL (VALUE (QUOTE primer-read-var)))
   (MEMBER (VALUE (QUOTE process-variables))))

(debug-loop
   (POSITION NIL)
   (EVAL try-loop))

(try-loop
   (POSITION (VALUE (4 . 144)))
   (IF (EVAL loop))
   (THEN (PRIN1 "works just fine"))
   (ELSE (EVAL using-read-var)))

(using-read-var
   (POSITION (VALUE (142 . 236)))
   (IF (MEMBER (fget #$read-next read-variable VALUE)
               (fget #$process process-variables VALUE)))
   (THEN (EVAL read-vars-equal))
   (ELSE (EVAL advise-user)))

(read-vars-equal
   (POSITION (VALUE (142 . 139)))
   (IF (EQUAL (fget #$primer-read primer-read-var VALUE)
              (fget #$read-next read-variable VALUE)))
   (THEN (EVAL altering-cond-var))
   (ELSE (EVAL advise-user)))

(altering-cond-var
   (POSITION (VALUE (144 . 56)))
   (IF (MEMBER (fget #$condition condition-test-var)
               (fget #$process process-variables)))
   (THEN NIL)
   (ELSE (EVAL advise-user)))

(advise-user
   (POSITION (VALUE (313 . 136))))
```