AI Magazine Volume 3 Number 4 (1982) (© AAAI)

A Representation System User Interface For Knowledge Base Designers

Richard E. Fikes

Cognitive and Instructional Sciences Group
Xerox Palo Alto Research Center
Palo Alto, California

A MAJOR STRENGTH of frame-based knowledge representation languages is their ability to provide the knowledge base designer with a concise and intuitively appealing means of expression. The claim of intuitive appeal is based on the observation that the object-centered style of description provided by these languages often closely matches a designer's understanding of the domain being modeled and therefore lessens the burden of reformulation involved in developing a formal description.

To be effective as a knowledge base development tool, a language needs to be supported by an implementation that facilitates creating, browsing, debugging, and editing the descriptions in the knowledge base. We have focused on providing such support in a SmallTalk (Ingalls, 1978) implementation of the KL-ONE knowledge representation language (Brachman, 1978), called KloneTalk, that has been in use by several projects for over a year at Xerox PARC. In this note, we describe those features of KloneTalk's display-based interface that have made it an effective knowledge base development tool, including the use of constraints to automatically determine descriptions of newly created data base items.

Viewing and Editing a Knowledge Base

KL-ONE provides a language for describing an inheritance network of *generic* and *individual concepts.* For the purposes of this discussion, a generic concept can be considered to be a description template ("frame," "unit") from which descriptions of individuals, in the form of individual concepts, are formed.

To initiate interaction with a knowledge base, some sort of index or "table of contents" is needed. In KloneTalk, we provide a *network index window* containing alphabetically ordered lists of names of the network's generic and individual concepts. A menu is available in an index window to perform network oriented operations such as "pretty printing" (as in LISP) a portion of the network onto text files. Also, the user can select a name in a list and obtain a menu of operations applicable to the selected concept. The operations include renaming, removing from the network, and viewing the concept.

Knowledge base design requires viewing and editing already existing descriptions. KloneTalk allows the user to select a concept in the network index window and request a view of it. Such an operation causes the appearance on the screen of a new window called a *concept viewer* (with a size and location specified by the user) containing a "pretty printed" description of the concept in a simple parenthesis language. Each generic concept in KL-ONE is considered to

Several people have contributed ideas and effort to KloneTalk. In particular, Austin Henderson was a co-initiator of the project, a major contributor to the system's design, and a participant in the programming Ron Brachman, Steve Weyer, and Ira Goldstein provided significant consulting support, and Ben Cohen participated in the programming.

```
                                  Explanation of Syntax
[Person                           Name of the concept.
   (Roles:
      (mother                     Name of the first role.
         (ValueIsA: Woman)        Constrains a Person's mother to be a Woman.
         (Number: 1))             Constrains a Person to have exactly 1 mother.
      (father                     .
         (ValueIsA: Man)          .
         (Number: 1))             .
      (child
         (ValueIsA: Person))      The number of children is unconstrained.
      (sex
         (ValueIsA: SexType)
         (Number: 1]
```

Figure 1. The "Person" generic concept

be a specialization of a collection of other generic concepts and is defined by describing a set of attributes (called roles) and a set of constraints (called role set relations) that must hold among the values of the attributes for any individuation of the concept Figure 1 shows an example of a view of a simple generic concept

Editing is done primarily by using the standard SmallTalk text editor to modify concept descriptions in concept viewers When the desired changes to a description have been made, a "compile" menu operation is available to replace the old version of the concept by the new one in the network

The definition of a concept in KL-ONE includes inherited information obtained from the concept's super-concepts and "local" information specific to the concept being defined. Only the local portion of the definition is editable, since the inherited information is a part of the definition of the super concepts However, the user often wants to see the entire definition in a view of a concept. Hence, some device is needed to distinguish for the user which parts of the definition are editable. In KloneTalk, the inclusion of inherited information in a view is optional, and when included, the local information is displayed in bold face to distinguish it from the inherited information. Figure 2 shows such a view that includes both the local and inherited information

The view of a concept provided in a concept viewer does not include all the information that a user might want to know about the concept. For example, it does not list the concept's individuals or specializations, nor does it list those roles of other concepts whose value restriction is the concept being viewed (The *value restriction* of a role is a generic concept, and it indicates that any value of the role must be an individuation of the generic.) To provide easy access to such commonly requested additional information, KloneTalk provides with each concept viewer a *viewer menu* whose information operations display their answers as lists of the names of the requested items.

Browsing a Knowledge Base

The knowledge base designer often wants to find out information about concepts that are related to a concept currently being viewed (i.e., to "browse" through the network) KloneTalk provides that capability by allowing the user to select the name of any concept mentioned in a view and then apply the operations on the viewer menu to the selected concept. For example, one could obtain a list of the specializations of the value restriction of one of the viewed roles Viewer menu operations include ones that display a view of the selected concept, so that the user can obtain a view of any other concept mentioned in the current view. The new view can optionally appear in the same window (displacing the existing description) or in a new concept viewer. Thus, for example, one could obtain a view of a concept that the current concept specializes.

To further facilitate browsing in KloneTalk, whenever a list of concept names is displayed by an information operation, any name on the list can be selected and the viewer menu then applies to the selected item. So, for example, the user can display a list of the specializations of a concept and then request a list of the individuals of one of those specializations

```
[Woman (aKindOf: Person)
   (Comment:  A Person whose sex is female.)
   (Roles:
      (mother (from: Person)
         (ValueIsA: Woman)
         (Number: 1))
      (father (from: Person)
         (ValueIsA: Man)
         (Number: 1))
      (child (from: Person)
         (ValueIsA: Person))
      (sex (from: Person)
         (Value: Female)
         (Number: 1]
```

Figure 2. The "Woman" generic concept with local information shown in bold.

Extending a Knowledge Base

We have found in using KloneTalk that a significant portion of our time is spent defining new concepts. Hence, we have focused on ways of easing that task. In this section we describe the facilities included in KloneTalk for that purpose.

Templates. The most primitive mechanism in the system is simply a menu-initiated operation for displaying a concept viewer containing a text template for a concept. The user can then edit the template and compile the resulting description in the network. The templates are particularly useful to a new or casual user who is unfamiliar with the details of the concept description language.

Specialization and Individuation. The most common method of defining new concepts is to use the network extension operations *specialize* and *individuate*, available on the viewer menu. Those operations prompt the user for the name of the specialization or individual, create an appropriate concept with that name if one is not already in the network, and then establish the appropriate specialization or individuation links. As is the case for all viewer menu operations, the concept being specialized or individuated is either the one being viewed or one whose name has been selected by the user.

Once a new specialization or individual has been created, the user is faced with the task of describing the new concept. A description consisting of the inherited information already exists for the new concept, and that description can provide a template-like context in which to specify the concept's distinguishing features by indicating the roles and constraints that are available for modification. The description task then becomes one of modifying and adding to (i.e., specializing) the existing description.

KloneTalk facilitates this style of description by having the viewer menu apply to the new concept after a network extension operation. So, for example, a user can specialize a concept and then obtain a view of the specialization in preparation for augmenting its definition via text editing operations.

Automatic definition of mentioned concepts and roles. During the editing of a concept description, one often wants to mention concepts and roles that have not yet been defined. The KloneTalk concept description compiler supports that process by automatically defining any item mentioned in a description that does not yet exist in the network and including in the description of the new item whatever information it has from the compilation context. So, for example, if a role's value is not already in the network, then it will be created as an individual concept individuating the role's value restriction.

Using constraints to augment descriptions The constraints included in generic concept descriptions provide information that can be used to determine parts of the description of individual concepts. The system can automatically add to the description of individuals whatever information it can deduce from such constraints, thereby freeing a user from the need to enter redundant information and protecting against the entry of information that violates the constraints.

We have implemented such a capability in KloneTalk for one frequently used form of constraint, called a role value map (RVM). An RVM constrains each individuation of a generic by specifying that one set of role values associated with that individual must be equal to or be a subset of a second set of role values associated with that individual For example, in the description of a "Parentage" given in

```
[Parentage
   (Comment: 'The relationship resulting from conceiving a child')
   (Roles:
      (mother
         (ValueIsA: Woman)
         (Number: 1))
      (father
         (ValueIsA: Man)
         (Number: 1))
      (child
         (ValueIsA: Person)
         (Number: (1,∞))))
   (RoleValueMaps:
      (ParentageSD1
         (Comment: 'The children of a Parentage are some of its father's children')
         (Map: (child from: Parentage) ⊆ (father from: Parentage) (child from: Person)))
      (ParentageSD2
         (Comment: 'The children of a Parentage are some of its mother's children')
         (Map: (child from: Parentage) ⊆ (mother from: Parentage) (child from: Person)))
      (ParentageSD3
         (Comment: 'The mother of a Parentage is each of its children's mother')
         (Map: (mother from: Parentage) = Each (child from: Parentage) (mother from: Person)))
      (ParentageSD4
         (Comment: 'The father of a Parentage is each of its children's father')
         (Map: (father from: Parentage) = Each (child from: Parentage) (father from: Person]
```

Figure 3. The "Parentage" generic concept

Figure 3, RVMs specify some of the relationships that must hold among the father, mother, and children whenever a child is conceived. To see how the system uses RVMs to add descriptive information to individuals, consider a situation where the user compiles the individuation of "Parentage" shown in Figure 4. Further, assume that the individual concepts "Sue," "Jack," and "Joan" did not previously exist in the network. The compiler will use the value restrictions of the Parentage roles to determine that "Sue" must be an individuation of Woman, "Jack" must be an individuation of Man, and "Joan" must be an individuation of Person. It will then conclude from the first RVM that "Joan" must be one of the values of the "child" role of "Jack," from the second that "Joan" must be one of the values of the "child" role of "Sue," from the third that "Sue" must be the value of the "mother" role of "Joan", and from the fourth that "Jack" must be the value of the "father" role of "Joan." The resulting descriptions are shown in Figure 5.

Thus, KloneTalk is able to define new individual concepts and supply a significant portion of their descriptions using the information it obtains from the associated generic descriptions.

Conclusions

KloneTalk has been effectively used in several activities at Xerox PARC, including designing the structure of data bases (Tou, 1982) and developing conceptual frameworks (Fikes 1981a).

Based on that experience, we can comment on some of the strengths and weaknesses of the facilities described in this note. The network index window needs to be augmented in some way to indicate the topological structure of the inheritance network. It seems that a simple node link graph would suffice, where the nodes denote concepts and the links denote specialization relationships. The user could then select nodes in this graph rather than names on the index lists.

Editing the concept descriptions as text has the usual problems of producing unbound parentheses and other syntax errors. A structure editor would be preferable. Although the use of bold characters works well as a way of distinguish

```
[ParentageOfJoan (a: Parentage)
  (Comment: 'The relationship resulting from conceiving Joan')
  (Roles:
    (mother (from: Parentage)
      (Value: Sue)
      (Number: 1))
    (father (from: Parentage)
      (Value: Jack)
      (Number: 1))
    (childJoan (Difs: child from: Parentage)
      (Value: Joan))
    (child
      (ValueIsA: Person)
      (Number: (1,∞))))
  (RoleValueMaps:
    (ParentageSD1
      (Comment: 'The children of a Parentage are some of its father's children')
      (Map: (child from: Parentage) ⊆ (father from: Parentage) (child from: Person)))
    (ParentageSD2
      (Comment: 'The children of a Parentage are some of its mother's children')
      (Map: (child from: Parentage) ⊆ (mother from: Parentage) (child from: Person)))
    (ParentageSD3
      (Comment: 'The mother of a Parentage is each of its children's mother')
      (Map: (mother from: Parentage) = Each (child from: Parentage) (mother from: Person)))
    (ParentageSD4
      (Comment: 'The father of a Parentage is each of its children's father')
      (Map: (father from: Parentage) = Each (child from: Parentage) (father from: Person]
```

Figure 4. An individuation of Parentage

ing local versus inherited information for the user, maintaining that distinction during editing can be a nuisance. That maintenance burden could be alleviated by a compiler that was sophisticated enough to determine which parts of the description differ from the inherited information and therefore need to be made local.

The "compile" operation that assigns a new description to an existing concept requires care in its design, particularly with respect to how the changes affect other concepts. For example, the description of a specialization of the compiled concept or a constraint in the description of any other concept may refer to a role that no longer exists in the newly compiled concept The current KloneTalk system assures syntactic consistency in the network by removing such references to deleted structures, but does not provide any other options to help with such problems. What seems to be needed is an interactive mechanism whereby the user is notified of each suspected anomoly and given the opportunity to specify what he wants done, perhaps as a selection among a set of standard options.

The most successful of the interface facilities have been those that allow a new concept to be described by editing the inherited description, that automatically define a concept or role when it is mentioned, and that use the RVMs to automatically fill in parts of a description More details on KloneTalk can be found in (Fikes, 1981b).

References

Brachman, R. et al KL-ONE Reference Manual, BBN Report No 3848, July 1978

Fikes, R A commitment-Based Framework for Describing Informal Cooperative Work *Proceedings of the Third Annual Conference of the Cognitive Science Society,* Berkeley, Calif August 1981 (to appear in the *Cognitive Science Journal*)

Fikes, R Highlights from KloneTalk: Display-Based Editing and Browsing, Decompositions, Qua Concepts, and Active Role Value Maps *Proceedings of the Second Annual KL-ONE Workshop* October 1981

Ingalls, D H , The Smalltalk-76 Programming System Design and Implementation *Conference Record of the Fifth Annual ACM Symposium on Principles of Programming Languages* January 1978.

Tou, F N. et al RABBIT: An Intelligent Database Assistant *Proceedings of the AAAI 1982 National Conference on Artificial Intelligence* August 1982, pp 314 18

```
[Joan (a: Person)
  (Roles:
    (mother (from: Person)
      (Value: Sue)
      (Number: 1))
    (father (from: Person)
      (Value: Jack)
      (Number: 1))
    (child (from: Person)
      (ValueIsA: Person))
    (sex (from: Person)
      (ValueIsA: SexType)
      (Number: 1]
```

Explanation of Syntax

```
[Jack (a: Man)
  (Roles:
    (mother (from: Person)
      (ValueIsA: Woman)
      (Number: 1))
    (father (from: Person)
      (ValueIsA: Man)
      (Number: 1))
    (sex (from: Person)
      (Value: Male)
      (Number: 1))
    (child (from: Person)
      (ValueIsA: Person)
      (Number: (1,∞)))
    (childJoan (Difs: child from: Person)
      (Value: Joan]
```

A *differentiation* of the "child" role, denoting one of Jack's children.

```
[Sue (a: Woman)
  (Roles:
    (mother (from: Person)
      (ValueIsA: Woman)
      (Number: 1))
    (father (from: Person)
      (ValueIsA: Man)
      (Number: 1))
    (sex (from: Person)
      (Value: Female)
      (Number: 1))
    (child (from: Person)
      (ValueIsA: Person)
      (Number: (1,∞)))
    (childJoan (Difs: child from: Person)
      (Value: Joan))]
```

Figure 5. Descriptions of individuals implied by ParentageOfJoan