"AN EXPERIMENT IN REPRESENTING THE KNOWLEDGE INVOLVED IN THE SPECIFICATION AND DESIGN OF SWITCHING SYSTEMS"

Jean-Francois CLOAREC, Jean-Francois CUDELOU

CNET LANNION-A SLC/ECA - 22301 LANNION - FRANCE

ABSTRACT

This experiment in representing knowledge has the goal of studying what kind of help existing knowledge representation tools might provide, for the process of designing switching systems. We offer some tentative positive as well as negative conclusions. The experiment is principally successful in that it has allowed us to point out some important representation problems, particularly in the checking and maintenance of consistency, and the expression of several points of view of a model.

I - INTRODUCTION

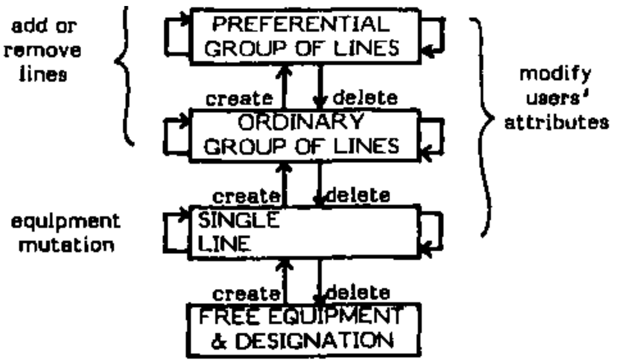
We have been addressing the task of representing and modeling the knowledge contained in a part of the general specifications of the French switching systems. These specifications, called "NEF" for : "Functioning and Exploitation Norms", give many sorts of informations, from the general missions of the system to the environment constraints and some basic know-how, all expressed informally in natural language.

The goal of this endeavor is to study the possibilities of bringing some help in the design process, especially for switching system software, mainly by providing some knowledge representation tools, suitable for all the stages of the design, and that fit a general method.

II - THE EXPERIMENT

A. Abstract model of the specifications

Representing the specifications as an "abstract model" is the first stage in the design process, so, we have first implemented a knowledge model related to a part of our specifications, the management procedure of the switching system, summarized by the following figure :



This procedure deals with objects, state transitions yielded by commands, and with consistency constraints.

So, the representation problems to cope with at this level are :

how to represent the objects ?
how to model their dynamic behavior and the man-machine commands ?
how to express the constraints on the model and the rules to check and enforce consistency ?
how to represent and deal with the multiple points of view of objects ?

B. Representation tool for the experiment; AIMDS

AIMD5 (Artificial Intelligence Meta Description System) is a knowledge representation system (N.S. SRIDHARAN [6]), that provides powerful representation schemes for our experiment : it allows generic objects and actions, instantiation of facts in a multi-contexts base, simulation of state changes with backtracking facilities, ability to describe definitional knowledge by means *of* meta-relations and to define consistency conditions used by the system to deduce facts and to check and/or restore the consistency of a model.

C. Building the model

We want to use only declarative descriptions so that we can translate easily and explicitly the non-procedural specifications, and obtain an extensible model easy to reason with. Nethertheless, this model must be executable so that we can demonstrate its validity by simulating its behavior and show that it tallies with the specified one.

It has been very easy to describe the objects of the model. For example, here is a part of the definition of the SINGLE-LINE object :

```
(TDN : ((SINGLE-LINE 0)
       ((ND = FN) DESIGNATION (INVERSE ND = OF FN)
            (COMPLEMENT NOT-ND = L))
       ((NE = FN) EQUIPEMENT...)
       ((NC = FN) COUNTER...)
       ((TY = L)  TYPES...)...))
```

This defines a generic SINGLE-LINE user by means of its three relations with a DESIGNATION object (e.g. phone number), with an EQUIPMENT object (physical wire), and with a COUNTER (account). The other relations are the ATTRIBUTES of the user. Each of these relations are given algebraic and deductive properties such as FN for functional 1 —> 1, L for list 1 —> n, or default value ; and meta relations like INVERSE between ND = and ND = OF or COMPLEMENT between ND = and NOT-ND =.

It has been easy also to describe the commands and the associated actions. For instance, the action to create a new SINGLE-LINE user is defined as follows :

```
(TDN : (  (ACTION-ABOCR A)
          ((ND = FN) DESIGNATION)
          ((NE = FN) EQUIPMENT)
          ((NC = FN) COUNTER (DEFAULT (LAMBDA (V)
              (FIND (A COUNTER (FREE YES))))))
          ((TY= L) TYPES)...))
```

With, for transition rules :

```
      (RULESFOR 'OPPORTUNITY)
((ACTION-ABOCR A (NC= C) (ND= D) (NE= E))
 (COUNTER C (FREE YES))
 (DESIGNATION D (FREE YES))
 (EQUIPMENT E (FREE YES)) )

      (RULESFOR 'PGOAL)
((ACTION-ABOCR A (NC= C) (ND= D) (NE= E) (TY= T)...)
(SINGLE-LINE NIL(NC= C)(NE= E)(ND= D)(TY= T)...))

      (RULESFOR 'OUTCOMES)
((ACTION-ABOCR A (NC= C) (ND= D) (NE= E)
 (COUNTER C (TAKEN YES))
 (DESIGNATION D (TAKEN YES))
 (EQUIPMENT E (TAKEN YES)) )
```

The opportunities are the preconditions to be checked before performing an action, the goals are the primary effects of the action, and the outcomes are the additional effects. TAKEN and FREE have been declared COMPLEMENT relations in the model.

These specifications permit AIMDS to simulate state changes.

D. The control structure

The control structure of the program is very simple, we just translate a given command into an AIMDS description and instantiate it as an ACTION. This produce, via PGOAL and OUTCOMES, the instantiation of the corresponding OBJECTS. If there is a failure in one of the action rules or in a consistency checking, an explicative message is issued and a backtrack performed.

E. The main representation difficulties

It has been far more difficult to solve the following problems :

1. How to deal with both consistency checking and consistency maintenance ? That is, what actions are permissible to restore consistency ?

We need both to express automatic enforcement of consistency, for example :

(A) TY = KLA ==> NOT-TY = CAD. That is to say that if a user had first a dial, and later is given a keyboard phone TY = KLA, his type dial TY = CAD must be changed to NOT-TY = CAD to restore consistency.

and strict consistency checking like :

(B) TY = MOD and TY = CAD ==> consistency failure. A user with a dial phone cannot have the modification privilege TY = MOD.

- First difficulty : A consistency condition (CC of AIMDS) used to maintain consistency has also failure effects.

For example, the condition (A) is expressed by the CC :

```
   (CC1) [(SINGLE-LINE X) NOT-TY = (TYPES CAD)
          IF (X TY = KLA)]
```

The value of the relation NOT-TY= is -checked by the value of the logical expression, and conversely, a change of the value of this expression yields to the according change of the relation. So, the assertion
(LINE-1 (TY= KLA)) with the previous value
(LINE-1 (TY= CAD)) leads to the correct result
(LINE-1 (TY= KLA) (NOT-TY= CAD)). But we need the symmetrical behavior expressed by the CC :

```
   (CC2) [(SINGLE-LINE X) NOT-TY= (TYPES KLA)
          IF (X TY- CAD)]
```

With these two CCs, the very same assertion will produce a failure because TY = KLA is forbidden by (CC2). So we can't express so easily the desired consistency maintenance.

The solution has been to express the consistency maintenance between an object and the action that modifies or creates this object. In fact, to avoid the repetition of the CCs for each action, we regroup their common attributes to be maintained in a single generic object, called M-ATTRIBUTE. The CCs become like :

```
  [(SINGLE-LINE X) NOT-TY = (TYPES CAD)
IF (X (SINGLE-LINE-OBJECTOF-M-ATTRIBUTE
      M-ATTRIBUTE-TY = -TYPES) KLA) 1
```

So, the consistency of the object is maintained by using the new asserted attribute values in the action, without any undesirable effect.

- Second difficulty : How to perform consistency maintenance first and then consistency checking ?

When we have both consistency checking cases like (B), and consistency maintenance like (A), we need to execute the checking after the maintenance. So, if the previous value is TY = CAD, when asserting TY = MOD + KLA we expect that first TY = KLA will enforce NOT-TY = CAD, so that TY = MOD will be accepted.

The solution has been to define a second generic object, called C-ATTRIBUTE, with the relations to be checked. The right order is assumed in instantiating M-ATTRIBUTE before C-ATTRIBUTE.

So, the final meta-model is :

```
                 (1)                 (1)
  COMMAND——>    ACTION———————>            OBJECT
                    I   1(2)              (2)      "
                  (3)   I—►TO-MAINTAIN-J  (3)
(i) = Sequence order 1    ►TO-CHECK
```

2. How to give the model some knowledge about its own generic structure ?

To reason on the structure, we need, at least, to know what kind of action is associated with a given command and which object is concerned.

But most of the representation languages, and this includes AIMDS, give us only facilities to reason upon facts and instantiated objects.

Our solution has been to describe the structural relations in each command with default values, for example in ABOCR :

```
((ACTION FN) TEMPLATE (DEFAULT ACTION-ABOCR))
((OBJECT FN) TEMPLATE (DEFAULT SINGLE-LINE))
```

So, it is easy to find the action and the object template-names related to a command. This is a specialized solution to this specific version of the problem. But the general problem of knowing, reasoning on, and modifying the generic structure of knowledge still remains.

3. How to represent distinct points of view of the model ?

The specifications address several functional aspects of systems, such as services to users, bill and management ; and the design process involves new aspects as electronics and layout. It is easy to represent these points of view in using several perspectives (KRL C 1 \ FRL [5], or super-concepts (KLONE [2], AKO of AIMDS). But, in fact, these representations would mix all the different aspects in the instantiated objects, via the inheritance mechanism. What we need, is to split the model in multiple views, to be able to reason sometimes in a subdomain, sometimes in another, without hampering ourselves with irrelevant aspects.

Our solution has been to define new meta-relations, one called SEMANTIC to jump from the objective view of constructed objects to a given subjective semantic (inverse SEMANTIC-OF) ; and one called ASPECT to jump from one point of view to another.

But we have still to study the properties of these relations and what could be the problems of consistency between all these different views.

III CONTINUATION OF THE EXPERIMENT

The continuation will have to cope with the problem of scaling-up and with new representation problems linked to the design process.

A. Scaling up ;

The present experiment included the generic definition of 12 commands, 12 actions and 20 objects, and in a typical simulation about 200 objects might be instantiated and 20 actions result from commands.

Although the full specification of the NEF commands would require 10 times more commands, it is our judgment, since we have experimented a representative sample, that scaling up will not reveal any new conceptual problem.

B. Design process and knowledge representation ;

The design process will have to manipulate both the top-down and bottom-up approaches. This involves : developing design alternatives, since there are always several choices to be had, from different functional organizations, to the use of different technologies. The context mechanism as in PIE [3], CONNIVER [43, or AIMDS could be a solution. This needs the definition of a distance measure between structural objects to find whether a solution could fit and with what conditions and constraint propagations.

We plan to tackle these problems in using our model to study the design of the corresponding software of the system management center.

VI CONCLUSIONS

This experiment allowed us to show some non-trivial problems in knowledge representation and to clarify what a good knowledge representation system could do for us.

We will emphasize upon :
- The need to deal with several points of view with the aim of reasoning and building objects in one or in others, and to bring back partial subjective solutions to be merged in an objective design.
- The greater importance for us to manipulate structural knowledge than facts. But these latter are nevertheless used to simulate dynamic behavior in validation phases.
- The tremendous importance of the consistency problem, where we need to deal both with checking and maintenance aspects, on generic knowledge as on facts.

But none of the existing knowledge representation systems seems to give solution to all these problems. Particularly, the multi-points of view approach is confused with the multiple inheritance mechanism ; and even in AIMDS, there is no clear separation between checking and maintenance of consistency. Futhermore, these representation tools are mainly applicable to the instantiated knowledge and are never concerned with the consistency of definitional, structural knowledge.

Nevertheless, the approach of AIMDS that allows objects and actions, multi-contexts and states, definitional knowledge by means of meta-relations, seems particularly powerful for its uniformity of representation and its possibilities in dynamic behavior representations.

ACKNOWLEDGMENTS

We thank RUTGERS University and especially N.S. SRIDHARAN, without whom this experiment would not have been possible.

REFERENCES

[1] D. Bobrow, T. Winograd : "An Overview of KRL Representation Language". Cognitive Science 1:1 1977.

[2] R.J. Brachman : "A Structural Paradigm for Representing Knowledge". BBN Report N° 3605 May 1978.

[3] I. Goldstein, D. Bobrow : "Representing Design Alternatives". CSL 81-3 March 1981, Xerox Park, Palo Alto Research Center.

[4] D. McDermott, G.L. Sussman : "The CONNIVER Reference Manual". AI Memo N° 259, May 1972, MIT.

[5] R.B. Roberts, LP. Goldstein : "FRL User's Manual". AI Memo N° 408 April 1977, MIT.

[6] N.S. Sridharan, B.L. Lantz, J.L. Bresina, J.L. Goodson : "AIMDS User's Manual". Vers. 3, Nov. 1981, RUTGERS University, New Brunswick (NJ).