From: AAAI-80 Proceedings. Copyright © 1980, AAAI (www.aaai.org). All rights reserved.

INTERACTIVE FRAME INSTANTIATION

Carl Engelman
Ethan A. Scarl
Charles H. Berg*

The MITRE Corporation
P.O.Box 208
Bedford, MA 01730

ABSTRACT

This paper discusses the requirements that interactive frame instantiation imposes on constraint verification. The representations and algorithms of an implemented software solution are presented.

INTRODUCTION

A number of frame representation languages or data access packages, seven of which are discussed in [STEFIK], have been developed as LISP extensions. In most applications of these languages, frame instantiation -- the creation of a new frame which represents an "instance", i.e., a more specific example, of a given "generic" frame -- occurs as a major theme. Yet, these languages do not really provide control structures sufficient to support interactive frame instantiation.

Most of this paper will be concerned with constraint verification. A frame representation language typically provides the programmer with a way of attaching a constraint as a "facet" of a given slot. It will reject any proposed values for the slot which fail that constraint, screaming a bit to the user. Such constraints attached to a slot in some generic frame also obtain automatically for slots of the same name occurring within its progeny. That is, they are "inherited". And that's about it. To explain why we felt the need for more control of constraint verification and, in fact, of the whole dynamics of interactive frame instantiation, we must present just a bit of our application.

THE APPLICATION

The KNOBS project [ENGELMAN] is directed towards the development of experimental consultant systems for tactical air command and control. We chose to focus first on what seemed to be a very simple type of aid. Imagine an Air Force officer is trying to plan a mission to strike some particular target. We are providing a program which interactively accepts the target, the airbase from which to fly the mission, the type of plane, the time of take-off, etc., and checks the input for inconsistencies and oversights. Such missions are stereotypes which are represented naturally as frames and the checks are constraints among the possible slot values in such frames.

*Current affiliation, AUTOMATIX Inc.

DATA BASE/LANGUAGE SETTING

We first translated FRL [ROBERTSJULY] [ROBERTSSEPT] from MACLISP to INTERLISP [ERICSON]. Data bases of targets and resources have been implemented as nets of individual and generic frames. An individual target frame, for example, contains information, e.g., location, specific to a particular target, while a generic target frame contains information true about classes of targets, for instance, the type of radar normally associated with a particular kind of surface-to-air missile. In all, the data base currently contains some 1400 frames.

We have introduced several upwards compatible extensions to FRL, e.g., programmer controlled parallel inheritance along paths defined by a specified set of slot names and the controlled automatic invocation of "$IF-NEEDED" procedures during attempts to retrieve missing data. We split the concept of generic frame: those which we continue to refer to as "generic frames" contain information (defaults, attached demons, etc.) applicable to all their instances. Their slots are not necessarily in correspondence with those of their instances. What we refer to as a "template", on the other hand, is a prototypical representation of an instance of the associated generic frame. Its slots correspond to those to be found in an instance, but contain "$IF-NEEDED" procedures where the instance would contain values. It also contains constraints on the values that may appear in an instantiation.

We also differentiate frames representing fully specified objects from those representing subclasses. The former is referred to as "instance" in [STEFIK] and [SZOLOVITS], and we call it "individual". Such a frame is identified by having an "AIO" (An-Individual-Of) slot pointing back to its generic frame. AIO corresponds to set membership, while AKO (A-Kind-Of) corresponds to set inclusion. While the frame instantiation procedures are designed for use at any level, we have thus far employed them only in the creation of individual frames.

DESIDERATA

Our goal is a demonstration which, like a good shortstop, makes it look easy. Some requirements are:

1. The system must know what information is needed and how to ask for it. It must also know when the instantiation is complete and what to do then.

2. The user must be able to enter a value for any slot in any frame at any time and the system must know what must be checked or rechecked and what remains to be done.

3. There should be a general facility to suggest choices for a slot which are consistent with the current values of the other slots.

4. The system should complain as soon as the slot values become inconsistent. It must show dynamic discretion in explaining enough, but not too much, of the difficulty.

5. The user must be able to ask questions about the data base and about the status of the instantiation.

6. The instantiation of one frame must be able to initiate the instantiation of another.

7. Constraint satisfaction must be maintained after the original instantiation whenever a slot value is changed or a template is changed.

REPRESENTATION

Templates

Templates are represented by frames with slots whose names will be replicated in the instantiated frames and which contain either $IF-NEEDED or $VALUE facets. The $IF-NEEDED procedures are responsible for deciding whether the value can be computed or is to be requested from the user. It is our normal practice to have the $IF-NEEDED procedures also perform type checking. The presence of a $VALUE facet causes a recursive call to the instantiator.

The template also contains two special slots, named CONSTRAINTS (discussed below) and BOOKKEEP. The BOOKKEEP slot contains procedures to be run upon completion of the frame instantiation, a sort of "IF-ADDED" mechanism at the frame level. The interpreter either steps through the template filling slots or fills those commanded by the user. Moreover, the user may interact at any time with LISP or with a natural language Q/A system for retrieving facts from the data base, including those inferred through "inheritance". The latter is implemented as an ATN parser, whose syntax and semantics are intimately related to the structure of the frame net. In addition, there are a number of amenities: spelling correctors, synonym and word truncation recognizers, and facilities for viewing the current status of the instantiation process, i.e., a presentation of the current slot values, distinguishing with explanation those which violate constraints.

Constraints

The form of a constraint is: (name domain expr), where "name" is an atom used to index a user-oriented explanation of the constraint, "domain" (the terminology is suggested in [STANSFIELD]) is the list of slot names whose interrelations are tested by the constraint, and "expr" is a predicate to be satisfied. "Expr" can refer to the current values of slots being instantiated simply by reference to their slot names. We define a bucket as an unordered list of constraints. The contents of the CONSTRAINTS slot in the template is a list of buckets, ordered to express priority. All constraints in the same bucket are of equal priority. The attachment of constraints to the template at the slot level, rather than the usual attachment to slots at the facet level, reflects our view that all the action is in the interaction of the slots and that it is presumptive to make a decision -- especially a static one -- as to which slot is "bad".

EXECUTION

Assuming a sequence of values is being suggested for the instantiated slots, the algorithm is as follows:

Initially all constraints are unmarked.

At any time, there is a current slot name, the one for which a value has been most recently proposed. A constraint is considered timely if its domain includes the current slot name and if all the other slot names in its domain have values already assigned. The interpreter passes through the buckets in decreasing priority until it discovers a timely constraint. If the constraint fails, the interpreter marks the constraint and traps the slots in its domain, i.e., renders their values unknown to constraints in lower priority buckets, which are not tested. If these lower priority constraints are already marked, they become unmarked since they are no longer timely. Other failed constraints in the current bucket are marked and their domains trapped. If a previously marked constraint now succeeds, then it is unmarked. If all the constraints in a bucket having a given slot name become unmarked, the slot name is released, i.e., pushed down to lower priority buckets along with the current slot name. The process normally terminates when all the slots are filled and none of the constraints are marked.

CONSISTENCY MAINTENANCE

Should a slot value in an instantiated frame or a constraint in a template be changed, the system makes appropriate checks.

COMPLEX CONSTRAINTS

The discussion above deals with constraints on the related slots in a given frame. Such constraints are called simple in [STANSFIELD]. What he calls complex constraints, those involving slots in different frames, are of great importance to us. For example, we must be concerned with the timing constraints needed to synchronize a primary mission with its support missions (refueling, defense suppression, etc.). We are currently engaged in the design and implementation of suitable representations and algorithms. We believe that a purely recursive (depth first) sequence of frame instantiations is not acceptable, and that we shall have to provide flexible control of interleaved "co-instantiations".

CHOICE GENERATION

When cueing the user for a slot-value, we would often like to present a list of values consistent with those already chosen for other slots. This is, in general, computationally impossible. It turns out, however, to be in the nature of our application that we frequently can produce a list of consistent values. Furthermore, we can do this by a fairly general method, generating the choices, in fact, from the constraints. The key point -- and this is obviously application dependent -- is that many of our constraints are of the form (name (A B1 -- Bn) (MEMBER A (foo B1 -- Bn))), where, if the code is to make sense, (foo B1 -- Bn) is a computable, finite list. So, for example, one constraint might mean:

(1) The airbase is one of our airbases in Europe.

and another might mean:

(2) The chosen fighter wing is located at the chosen airbase.

We say a constraint enumerates a slot, S, absolutely iff it is of the form: (name (S)(MEMBER S----)). Constraint (1), above, enumerates airbases absolutely. A constraint enumerates a slot, S, relative to slots S1, ---, Sn iff it is of the form: (name (S S1 --- Sn) (MEMBER S ---)). Constraint (2), above, enumerates the fighter wing slot relative to the airbase slot. The idea is to try to construct an acyclic graph whose nodes are slot-constraint pairs, such that the constraint at a node enumerates the slot at that node relative to those of the nodes pointed to by the arcs leaving the node. If the graph has a single source node S and all the terminal nodes represent absolute enumerations of their associated slots, then there is an algorithm which can enumerate the values for S, consistent with the already chosen slot values.

To make choice generation more efficient, we optimize the constraints within the current context by collecting those which are timely and "compiling" them into a function of only the current slot, essentially by pre-evaluating all subexpressions which do not contain this slot.

CRITICISM AND FUTURE DIRECTIONS

1) We need to design and implement a comparable system for the complex constraints.

2) The only relative priorities we can express between constraints are static. This might, someday, prove inadequate.

3) There is danger of an existential trap. The simplest example occurs when the first slot filled is A and the candidates value satisfies every constraint whose domain is (A). There may, however, be a constraint whose domain is (A B) which cannot be satisfied with the proposed value of A and any value of B. Our interpreter does not see this until B is selected. The choice generation scheme discussed above could also be employed to test (perhaps very expensively), for such traps.

ACKNOWLEDGEMENTS

This work was supported by the Rome Air Development Center under Air Force contract F19628-80-C-0001.

The FRL system was originally created at MIT by R. Bruce Roberts and Ira P. Goldstein. Roberts assisted us by defining and creating an "export nucleus" of FRL. The work reported here is built directly on the INTERLISP version of FRL translated and extended by Lars W. Ericson, while he was at MITRE. We are deeply indebted to him.

REFERENCES

[ENGELMAN] Engelman, C., Berg, Charles H., and Bischoff, Miriam, "KNOBS: An Experimental Knowledge Based Tactical Air Mission Planning System and a Rule Based Aircraft Identification Simulation Facility", Proc. Sixth Inter. Joint Conf. Artificial Intelligence, Tokyo, 1979, pp. 247-249.

[ERICSON] Ericson, Lars W., "Translation of Programs from MACLISP to INTERLISP", MTR-3874, The MITRE Corporation, Bedford, MA, Nov. 1979.

[ROBERTSJULY] Roberts, R. Bruce, and Goldstein, Ira P., "The FRL Primer", MIT AI Lab. Memo 408, July 1977.

[ROBERTSSEPT] Roberts, R. Bruce, and Goldstein, Ira P., "The FRL Manual", MIT AI Lab. Memo 409, September 1977.

[STANSFIELD] Stansfield, James L., "Developing Support Systems for Information Analysis", in Artificial Intelligence, An MIT Perspective, Winston, P. H., and Brown, R. H., (Eds.), The MIT Press, Cambridge, MA, 1979.

[STEFIK] Stefik, Mark, "An Examination of a Frame-Structured System", Proc. Sixth Inter. Joint Conf. on Artificial Intelligence, Tokyo, 1979, pp. 845-852.

[SZOLOVITS] Szolovits, P., Hawkinson, L. B., Martin, W. A., "An Overview of Owl, A Language for Knowledge Representation", MIT/LCS/TM-86, MIT, Cambridge, MA, June, 1977.