# STROBE: SUPPORT FOR STRUCTURED OBJECT KNOWLEDGE REPRESENTATION

*Reid G. Smith*

Schlumberger-Doll Research
Old Quarry Road
Ridgefield, Connecticut 06877

**ABSTRACT**

STROBE is a system that provides object-oriented programming support tools for INTERLISP. It offers a primitive foundation with which more complex structured object representation schemes can be constructed. STROBE implements multiple resident knowledge bases, tangled generalization hierarchies, flexible inheritance of properties, procedural attachment, and event-sensitive procedure invocation.

## 1. INTRODUCTION

The goal of designing second generation expert systems that are able to reason from causal models as well as the familiar compiled expertise encoded in rules is dependent on flexible tools for representing knowledge. In recent years there has been considerable interest in *structured object representation.* Within this framework, a programmer can encapsulate *packets* of knowledge and link them together via a variety of relationships to form knowledge bases. Inheritance of properties through generalization hierarchies is standard.

A problem facing the representation system designer is that many decisions he might make are implicitly biased toward a particular problem domain. The result is a system that may be too brittle to be easily applied to other domains. One approach to the problem is a low-level system that imposes a flexible structure and provides tools that allow a user to adjust the operation of the system and to embed it in higher-level systems that offer increased structure.

This is the motivation for STROBE, a low-level system for support of structured objects in INTERLISP. It is to be viewed more as an augmentation to INTERLISP that simplifies programming with structured objects than as a powerful representation system. In this light, major concerns in its design have been flexibility and efficiency. Mechanisms are provided through which a user can encode in his knowledge bases explicit information to adjust the skeletal STROBE structure in a manner specific to particular applications.[1]

## 2. STROBE: OVERVIEW

A STROBE knowledge base is a collection of *objects* whose characteristics are elucidated through a number of *slots.* Alternatively it can be viewed as a semantic network of nodes and unidirectional links.

*(DEFOBJECT <object> <type>*
*<generalizations> <groups> <slots>)*

constructs a new *<object>* whose *<type>* is one of the standard STROBE object types (e.g., *class* or *individual*). *<generalizations>* is one or more objects of which *<object>* is to be a specialization and from which it will inherit properties. *<groups>* is one or more collections of objects to which *<object>* should be added. Groups are not related to generalization hierarchies. *<slots>* is a list of object descriptors to be initially associated with *<object>*. Additional slots may be dynamically defined. Each slot is specified by a set of *facets* as follows.

*(<name> (<facetname$_1$> . <facetvalue$_1$>) ...*
*(<facetname$_n$> . <facetvalue$_n$>)).*

The *DUNE* object below is defined to be a specialization of *COASTAL-BARRIER-ISLAND.* It denotes a *CLASS* of objects as opposed to a particular example of a *COASTAL-BARRIER-ISLAND.* It is defined to be a member of the *SILICICLASTIC-ROCKS* and *GEOLOGY* groups. Three slots are defined: *PATTERN, CROSS-SECTION,* and *PLAN-VIEW.*

```
(DEFOBJECT DUNE CLASS COASTAL-BARRIER-ISLAND
  (SILICICLASTIC-ROCKS GEOLOGY)
  ((PATTERN (DATATYPE . OBJECT)
            (VALUE . GREEN-OVER-BLUE))
   (CROSS-SECTION (DATATYPE . BITMAP))
   (PLAN-VIEW (DATATYPE . BITMAP))))
```

STROBE defines a *value* facet for every slot. This facet effects the linkage from one node in the network to another. The contents of the *value* facet is the node (or list of nodes) pointed to by the slot. For *DUNE*, the value of the *PATTERN* slot is to be initially filled with the object *GREEN-OVER-BLUE.*

A *datatype* facet is also defined for every slot. Not all nodes in a STROBE knowledge base need be objects. They may be LISP functions, S-expressions, bitmaps, arrays, and so on. These nodes have the characteristics that: *(i)* they have no additional STROBE structure, and *(ii)* they are *leaf* nodes--they do not point to any other nodes. The *datatype* facet of a slot points to an object that specifies how the node pointed to by the *value* facet is to be interpreted.[2] In the *DUNE* object, the *value* facet of *PATTERN* is to be filled with an object. The *value* facets of *CROSS-SECTION* and *PLAN-VIEW* are to be filled with bitmaps.

1. RLL [Greiner, 1980] has a similar flavor but emphasizes self-description over efficiency. GLISP [Novak, 1982] places more emphasis on efficiency and less on flexibility.

2. This idea was used in UNITS [Stefik, 1979].

Figure 1 shows the screen of the Xerox 1100 running the STROBE editor.[3] *BARCHAN-DUNE,* a specialization of *DUNE,* is shown. It has inherited the slots of *DUNE* and the *value* facets of the *CROSS-SECTION* and *PLAN-VIEW* slots have been filled with bitmaps.

## 3. INHERITANCE

If necessary, STROBE performs a run-time breadth-first search through the ancestors of an object to find the contents of a facet of a slot in that object.[4] The ancestors actually only provide a default for inheritance. Every STROBE slot access function allows the user to specify a partial path of objects to use as starting points for a search for inherited properties. This generality enables construction of objects that inherit slots from objects that are linked to them by relationships other than generalization (e.g., parts).

## 4. PROCEDURAL ATTACHMENT AND INDIRECT PROCEDURE INVOCATION

A procedure may be associated with a slot facet and invoked indirectly by sending a STROBE message to that facet. For example, a message sent to the *value* facet of the *BUILD* slot of a *BOX* object will cause the *BUILDBOX* procedure to be invoked. Many other objects may have *BUILD* slots--filled with procedures specific to the objects themselves.

*(DEFOBJECT BOX CLASS (GEOMETRIC-STRUCTURE ICON) NIL*
*((BUILD (DATATYPE . LISP)*
*(VALUE (LAMBDA (OBJECT SLOT FACET WIDTH HEIGHT)*
*(BUILDBOX WIDTH HEIGHT))))))*

If the facet to which the message is addressed is not found in the slot of the object (even after an inheritance search), then STROBE reroutes the message to the object that is the datatype for the slot. This indirection enables a programmer to encapsulate information about how to deal with actions that are generic to a datum of a particular datatype.

## 5. EVENT-SENSITIVE PROCEDURE INVOCATION

STROBE checks for procedures to be invoked whenever one of a number of significant *events* occurs. This gives a user considerable freedom to adjust the basic mechanisms of the system in ways that are specific to particular objects or classes of object.

**Object Creation/Deletion Procedures:** Invoked after an object has been created or before an object is deleted. They can be used to perform specialized initialization or cleanup. For example, when an instance of *TEST* is created, *FillTESTSlots* is invoked to initialize its slots. Upon deletion, *SummarizeTESTSlots* is invoked to summarize information contained in the slots before it is lost.

*(DEFOBJECT TEST CLASS ROOT NIL*
*((OBJECT-CREATION-PROCEDURES*
*(DATATYPE . LISP)*
*(VALUE FillTESTSlots))*
*(OBJECT-DELETION-PROCEDURES*
*(DATATYPE . LISP)*
*(VALUE SummarizeTESTSlots))))*

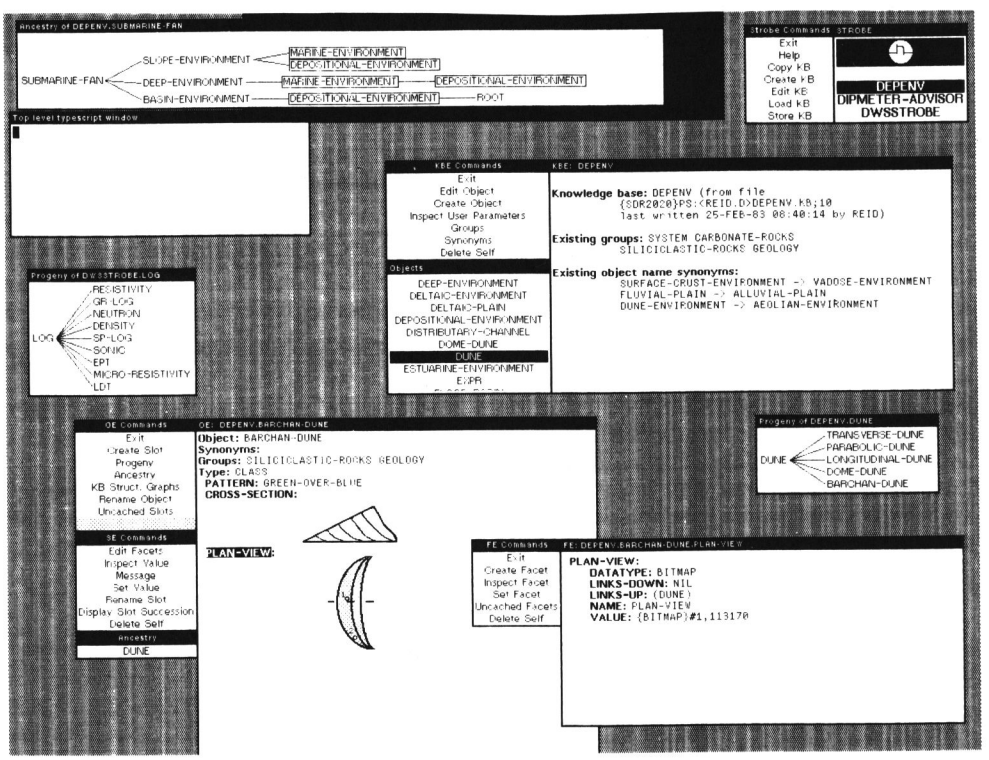


**Figure 1.** STROBE Interactive Editor Windows

3. The editor was written by Eric Schoen.

4. STROBE provides mechanisms that can be used to increase efficiency (e.g. caching slot pointers).

**Slot Creation/Deletion Procedures:** Invoked after a slot has been created or before a slot is deleted. In the following example, the functions *AddObjectToSlot* and *RemoveObjectFromSlot* maintain an association list that indexes a knowledge base by slot name. (STROBE provides a mechanism that enables a user to store such a data structure with a knowledge base.)

```
(DEFOBJECT INDEX CLASS ROOT NIL
 ((SLOT-CREATION-PROCEDURES
   (DATATYPE . LISP)
   (VALUE AddObjectToSlot))
  (SLOT-DELETION-PROCEDURES
   (DATATYPE . LISP)
   (VALUE RemoveObjectFromSlot))))
```

**Slot Access/Alteration Procedures:** Invoked *before* and *after* every attempt to access or alter the value of a slot. In the following example, the value of the *AREA* slot is computed from the *WIDTH* and *LENGTH* slots via the procedure found in the *ACCESS-PROCEDURES* facet. The keyword *AFTER* indicates that the procedure should be fired *after* the value has been retrieved via lookup. *SOURCE-OBJECT* is the object in which the slot was found (perhaps via inheritance), and *VAL* is the value retrieved via lookup (ignored in this example). *GETVALUE* is a STROBE function for accessing the value of a slot.[5]

```
(DEFOBJECT COMPOSITE CLASS ROOT NIL
 ((WIDTH (DATATYPE . EXPR) (VALUE . 3))
  (LENGTH (DATATYPE . EXPR) (VALUE . 2))
  (AREA (DATATYPE . EXPR)
        (ACCESS-PROCEDURES
         AFTER
         (LAMBDA (OBJECT SOURCE-OBJECT SLOT VAL)
              (ITIMES (GETVALUE OBJECT 'WIDTH)
                      (GETVALUE OBJECT 'LENGTH)))))))
```

In the following example, the value of the *S1* slot can only be filled with *BITMAP* or *EXPR*. This is accomplished with a *RESTRICTIONS* facet and an alteration procedure that enforces the restriction. (If a procedure fired *before* the new value is placed returns **FAIL**, then placement of the new value is prevented.) *VAL* is the new value to be placed. *GETFACET?* is a STROBE function for accessing a facet.

```
(DEFOBJECT R1 CLASS ROOT NIL
 ((S1 (DATATYPE . OBJECT) (RESTRICTIONS BITMAP EXPR)
      (ALTERATION-PROCEDURES
       BEFORE
       (LAMBDA (OBJECT SOURCE-OBJECT SLOT VAL)
         (COND ((FMEMB VAL
                  (GETFACET? OBJECT SLOT 'RESTRICTIONS))
                VAL)
               (T '*FAIL*)))))))
```

In the following example, the *before* procedure associated with the *PARTS* and *PARTOF* slots of *T1* redefines the operation of placing a value in those slots. The new operation adds an object without duplication to a list of objects. It also sets a variable (*NewValue*) for use by the *after* procedure. That procedure sets up a back pointer. If object *Part1* is added to the *PARTS* slot of object *T1*, then *T1* is added to the *PARTOF* slot of *Part1*. Note that this kind of symmetric operation must be carried out *after* the new value has been placed to avoid infinite regress. It demonstrates the utility of both *before* and *after* procedures.[6]

```
(DEFOBJECT T1 CLASS ROOT NIL
 ((PARTS (DATATYPE . OBJECT)
   (ALTERATION-PROCEDURES
    (BEFORE
     (LAMBDA (OBJECT SOURCE-OBJECT SLOT VAL)
      (COND ((NULL (SLOTVALUEP OBJECT SLOT))
             (SETQ NewValue VAL) (LIST VAL))
            ((FMEMB VAL (GETVALUE OBJECT SLOT))
             (SETQ NewValue '*FAIL*))
            (T (SETQ NewValue VAL)
               (CONS VAL (GETVALUE OBJECT SLOT))))))
    (AFTER
     (LAMBDA (OBJECT SOURCE-OBJECT SLOT VAL)
      (COND ((NEQ NewValue '*FAIL*)
             (PUTVALUE NewValue 'PARTOF OBJECT)))))))
  (PARTOF (DATATYPE . OBJECT)
   (ALTERATION-PROCEDURES
    (BEFORE ... <identical to PARTS procedure> ... )
    (AFTER
     (LAMBDA (OBJECT SOURCE-OBJECT SLOT VAL)
      (COND ((NEQ NewValue '*FAIL*)
             (PUTVALUE NewValue 'PARTS OBJECT)))))))))
```

*SLOTVALUEP* returns NIL if no value has been set for a slot. *PUTVALUE* is a STROBE function for setting the value of a slot.

## 6. OTHER STROBE FEATURES

- **Multiple Resident Knowledge Bases:** A user may have several knowledge bases in memory at the same time.
- **Synonyms:** All STROBE functions resolve references to synonyms for objects and slots.
- **Instantiation from Complex Descriptions:** STROBE uses the *description* object type to specify templates for instantiating *class* objects. When an object whose slots contain descriptions is instantiated, STROBE creates new objects for each of the descriptions and resolves inter-object references.
- **Standard INTERLISP Source Files:** STROBE is integrated with the INTERLISP file package and constructs knowledge base files in standard source file format.
- **Object-Centered Memory Management:** A user can selectively page objects to secondary storage.[7]

5. In general, a list of procedures may be invoked on slot access or alteration.

6. The LOOPS [Bobrow, 1982] *active value* effectively corresponds to the STROBE *before* alteration procedure. There is no direct correspondence to the *after* procedure.

7. This is useful on a limited address machine like the DEC-20. It has not yet been needed on the Xerox 1100.

## 7. APPLICATIONS

STROBE has been used to structure information about data and control flow for two large well-log interpretation programs. The system has also been used to construct a two-dimensional equation prettyprinter as part of the interface to an automatic programming system for well-log interpretation software. In future it will be used to represent the domain knowledge, software knowledge, and intermediate stages of program development.

STROBE has been used as a rapid prototyping tool in the development of graphics facilities by Schlumberger's Houston Interpretation Engineering staff. These facilities are intended to generate the graphics for all commercial Schlumberger products.

Finally, as an experiment in adjusting the STROBE skeleton in the manner shown in this paper, a basic implementation of the class and inheritance structures of Smalltalk-80 was developed. It required the definition of a small number of objects, slots, and attached procedures and was completed in a few days.[8]

**Acknowledgements:** The design of STROBE has benefited greatly from its use by Stephen Smoliar, Eric Schoen, David Barstow, Roger Duffey, and Scott Marks. Gilles Lafue provided a number of helpful comments on this paper. The local network support provided by Stanley Vestal, Eric Schoen, and Ed Dolph has simplified the workstation implementation. The intellectual stimulation of object-oriented programming discussion group and the managerial support of James Baker are also appreciated.

## REFERENCES

D. G. Bobrow and M. J. Stefik, *A Virtual Machine for Experiments in Knowledge Representation.* Unpublished Memorandum, Xerox Palo Alto Research Center, April 1982.

R. Greiner and D. B. Lenat, A Representation Language Language. *Proceedings of the National Conference on Artificial Intelligence,* August 1980, pp. 165-169.

G. S. Novak, GLISP: A High-Level Language For A.I. Programming. *Proceedings of the National Conference on Artificial Intelligence,* August, 1982, pp. 238-241.

M. J. Stefik, An Examination Of A Frame-Structured Representation System. *Proceedings of the Sixth International Joint Conference On Artificial Intelligence,* August 1979, pp. 845-852.

8. This was done by David Barstow.