SELF: The Power of Simplicity*

DAVID UNGAR†
(ungar@self.stanford.edu)

Computer Systems Laboratory, Stanford University, Stanford, California 94305

RANDALL B. SMITH†
(rsrnith@parc.xerox.com)

Xerox Palo Alto Research Center, Palo Alto, California 94304

Abstract. SELF is an object-oriented language for exploratory programming based on a small number of simple and concrete ideas: prototypes, slots, and behavior. Prototypes combine inheritance and instantiation to provide a framework that is simpler and more flexible than most object-oriented languages. Slots unite variables and procedures into a single construct. This permits the inheritance hierarchy to take over the function of lexical scoping in conventional languages. Finally, because SELF does not distinguish state from behavior, it narrows the gaps between ordinary objects, procedures, and closures. SELF's simplicity and expressiveness offer new insights into object-oriented computation.

To thine own self be true.
—William Shakespeare

1 Introduction

Object-oriented programming languages are gaining acceptance, partly because they offer a useful perspective for designing computer programs. However, they do not all offer exactly the same perspective; there are many different ideas about the nature of object-oriented computation. In this paper, we present SELF, a programming language with a new perspective on objects and message passing. Like the Smalltalk-80¹ language [6], SELF is designed to support exploratory programming

---

*This work is partially supported by Xerox, and partially by National Science Foundation Presidential Young Investigator Grant #CCR-8657631, Sun Microsystems, the Powell Foundation, IBM, Apple Computer, DEC, NCR, Texas Instruments, and Cray Laboratories.

†Authors' present address: Sun Microsystems, 2500 Garcia Avenue, Mountain View, CA 94043.

This paper is a substantial revision of [20], originally published in OOPSLA '87 Conference Proceedings (SIGPLAN Notices, 22, 12 (1987) 227-241).
and therefore includes runtime typing (i.e. no type declarations) and automatic storage reclamation.

But unlike Smalltalk, SELF includes neither classes nor variables. Instead, SELF has adopted a prototype metaphor for object creation [2, 3, 4, 8, 10]. Furthermore, while Smalltalk and most other object-oriented languages support variable access as well as message passing, SELF objects access their state information by sending messages to "self," the receiver of the current message. Naturally this results in many messages sent to "self," and the language is named in honor of these messages.

One of the strengths of object-oriented programming lies in the uniform access to different kinds of stored and computed data, and the ideas in SELF result in even more uniformity, which results in greater expressive power. We believe that these ideas offer a new and useful view of object-oriented computation.

Several principles have guided the design of SELF:

**Messages-at-the-bottom.** SELF features message passing as the fundamental operation, providing access to stored state solely via messages. There are no variables, merely slots containing objects that return themselves. Since SELF objects access state solely by sending messages, message passing is more fundamental to SELF than to languages with variables.

**Occam's razor.** Throughout the design, we have aimed for conceptual economy:
- As described above, SELF's design omits classes and variables. Any object can perform the role of an instance or serve as a repository for shared information.
- There is no distinction between accessing a variable and sending a message.
- As in Smalltalk, the language kernel has no control structures. Instead, closures and polymorphism support arbitrary control structures within the language.
- Unlike Smalltalk, SELF objects and procedures are woven from the same yarn by representing procedures as prototypes of activation records. This technique allows activation records to be created in the same way as other objects, by cloning prototypes. In addition to sharing the same model of creation, procedures also store their variables and maintain their environment information the same way as ordinary objects, as described in Section 4.

**Concreteness.** Our tastes have led us to a metaphor whose elements are as concrete as possible [14, 15]. So, in the matter of classes versus prototypes, we have chosen to try prototypes. This makes a basic difference in the way that new objects are created. In a class-based language, an object would be created by instantiating a plan in its class. In a prototype-based language like SELF, an object would be created by cloning (copying) a prototype. In SELF, any object can be cloned.

---

1Smalltalk-80 is a trademark of ParcPlace Systems. In this paper, the term “Smalltalk” will be used to refer to the Smalltalk-80 programming language.