Originally published in *PIE Developers* magazine, July 1994. For information about PIE Developers, contact Creative Digital Systems at CDS.SEM@applelink.apple.com or 415-621-4252.

SELF and the Origins of NewtonScript

Walter Smith
Apple Computer, Inc.
wrs@apple.com

This is an introduction to the paper "Self: the power of simplicity", by Ungar and Smith.

When I joined the Newton group--then called the Special Projects Group--in 1988, it was still a research organization. The product, whatever it might be, was a long-term goal, and we had the now-incredible luxury of thinking about abstract problems and filling our file cabinets with papers ordered from the Apple Library document service.

One of the problems that interested us in those days was choosing a language for Newton. We knew it would be a big influence on the feel of the system, and it would have to be powerful and productive if we were going to do all the things we wanted. Object-oriented languages were just starting to enter the mainstream (even C++ was not yet the industry darling it is today), and there were a lot of interesting new languages to learn about and evaluate. In hindsight, actually, we became a little too distracted by choosing a language; this period of Newton history is not-so-fondly remembered as the "language thrash".

SELF stood out among the languages we saw during that period. Although it was clearly impractical as a system language (it took several years for the SELF compiler to reach its present impressive state), its audacious simplicity made an impression that the more "serious" languages did not. During the language thrash, as we conducted a series of objective evaluations that showed we should really be using Smalltalk, or Eiffel, or Trellis-Owl, or even C++, the conversation would often wishfully turn to SELF.

I had no idea I would end up designing a language for Newton. In fact, at that time we consciously avoided the temptation to design a Newton-specific language. Leave language design to the experts, we thought, and we'll just choose one.

To make a very long story short, it didn't happen that way. Reality set in, and during the development effort that brought you the MessagePad, a new language--now called NewtonScript--evolved in parallel with the view system and object store. The language thrash made it possible: all those languages we looked at provided a wealth of ideas that found their way into NewtonScript. SELF was one of the primary influences.

SELF and NewtonScript

As a NewtonScript user (which I assume you are or will be shortly), you may not immediately recognize SELF. NewtonScript is not a direct derivative. However, SELF inspired some of the most important aspects of NewtonScript.

The prototype-based inheritance system of NewtonScript is adapted from SELF's "parent slots"; it's more complex than the version of SELF described here, but simpler than some later versions. Newton's view system evolved in parallel with NewtonScript, and the idea of combining container inheritance defined by the view hierarchy with refinement in the form of view templates was naturally reflected in the language as "double inheritance". Later versions of SELF allowed any number of prioritized parent slots, but NewtonScript fixed the number and priority in the form of _parent and _proto slots.

The advantages of prototype-based inheritance are described somewhat abstractly in this paper. As a user of Newton Toolkit, you experience them frequently. When you draw a view in NTK and attach slots to it, you're not specifying parameters that will be read out of a resource file at runtime to generate an instance of a class that you specified somewhere else; you're constructing an actual object that will be embedded in your application and used directly at runtime. Eliminating that layer of abstraction gives Newton programming an unusually direct feel. "One-of-a-kind" objects are painful to create in a class-based system, as Ungar and Smith point out, and graphical user interfaces are full of them.

SELF combines variables, slots, and messages into one mechanism, but NewtonScript doesn't go quite so far. In SELF, you can't tell if referencing a variable will execute a method or just get a value from a slot. In NewtonScript, variable accesses are always simple slot accesses. If you want to execute a method, you have to send a message. There's no fancy theoretical explanation for this decision; I was just being conservative with space,time, and implementation complexity. The feature is sufficiently useful that it found its way into the system anyway, in the form of the SetValue function.

SELF's use of regular objects as activation records is a powerful idea. The NewtonScript documentation avoids such a direct description of the calling mechanism because we find very few people are interested in such details, and in fact the demands of speed draw the implementation away from such a direct approach, but here you can see the elegance of the original concept. Its original implementation in the NewtonScript interpreter may well have been smaller than its explanation.

It will be obvious that none of SELF's syntax was used in NewtonScript. Smalltalk syntax is elegant, but unfamiliar to most programmers. NewtonScript's Pascal-like (or, to give credit where due, Algol-like) syntax is easier to learn for many, with little loss of expressiveness and power.

Modern SELF

Active development continues on SELF, so if you find it interesting, there is more you will probably want to read. If you have access to the Internet, visit the SELF archive for the latest information on SELF.

Walter Smith joined the Newton group in 1988. He is the principal designer and implementor

of NewtonScript and the Newton object store.

wrs@apple.com, 8 July 1994