XEROX
PALO ALTO RESEARCH CENTER
Learning Research Group
April 25, 1978

To: Smalltalk Interest

Subject: NoteTaker Smalltalk Design ***DRAFT***

From: Dan Ingalls

Filed on: MAXC<Ingalls>NoteTalk.Bravo

General Approach
The NoteTaker Smalltalk design builds on that of its predecessor, with major changes addressed to the following improvements:

1. Full capacity performance in NoteTaker environment. It should be possible to do complex editing of combined text and graphics with 128K words of RAM and a 340K byte floppy disk.

2. Identical compatibility with the D-0 environment. It should be possible to run the same save file on either machine, except in cases where advantage has been taken of the greater virtual address space of the D-0.

3. Better real-time response through more efficient addressing, storage management and elimination of object swapping.

4. Support for language improvements such as multiple superclasses and arbitrary instantiation.

Pointer format
Our experience has shown 16-bit object pointers to be well-suited to a system size of 1 million words. NoteTaker Smalltalk will not encode class information in the pointer as in its predecessor for three reasons:

1. It causes inefficient use of address space for classes with few instances.

2. It makes classes very special, and thus requires an extra level of simulation to provide arbitrary instantiation.

3. It makes arbitrary transmutation of objects effectively impossible.

Physical Storage Management
There are two principal areas of physical storage which are called the ROT (Resident Object Table) and the data area. The ROT is indexed by object, and provides the actual core address (24 bits) and also the reference count of the object. The data area contains the remaining representation of the objects, namely class link and parts. The tradeoffs in space are interesting:

128K NoteTaker	approx. 10K objects
512K NoteTaker	approx. 40K objects
D-0 (1-2M paged)	limit 60K objects

In 128K, only 1 bit of 8 are being used for the high-order core address; one could save 10K bytes by going to 16-bit core addresses which get multiplied by two in the interpreter. Note that this evenword alignment gives rise to a mean waste of 1/2 word per object, or 10K again! The argument tips much in favor of long core addresses for the larger sizes, and it would be nice for the small system to be totally compatible.

There are two more places where we could pick up a little space. One is to pack a small reference count in with the high-order bits of the core address; the 8086 can only use 4 of these bits, but it would restrict D-0 to the same range (insofar as we choose to retain total compatibility). We could also restrict the object numbers of class-like objects to have zero in their low order bits so that a small reference count could be packed into those bits of the object's class link. This scheme implies that any future schemes for arbitrary instantiation will involve an indirect ancestor link, as the ancestor can not be counted on to have such a special object number. At this time I feel that this issue is a small one - 5K out of 128K, and we should cling to simplicity as long as possible.

It must be possible to trade ROT allocation off against the data area, but a proportion of 1 ROT entry per 16 words of data will probably do well initially.

The current allocation scheme for resident data seems appropriate for the NoteTaker as well: a set of free lists for the common sizes and octaves for large ones, a minimum quantum of two words (this may not be necessary), and a compaction pass for consolidation of fragments.

Virtual Storage Management (garbage collection)
Our fall-back position here is to do reference counting as now, and provide a garbage collector which would take 5 or 10 minutes to collect cyclic structures.

Ted has a new design for a composting scheme (special treatment for recently created objects) in the works which may reduce the reference count overhead. He is currently taking statistics on object activity, and I am currently writing code for reference counting on the 8086, and we will choose our strategy when the results are available.

Program encoding
The Smalltalk-76 instruction set has stood up very well, but a few improvements have emerged from our experience and should be incorporated in the NoteTaker Smalltalk:

1. Peter's recommendation for saving 2 bytes of overhead on methods.

2. Make Method a real Class to clean up the semantics of code.

3. Put all messages in-line with a send-immediate code, replacing the send-literal-relative code. This saves a byte per message for single occurrences - the dominant case. We would use 1 hexade of codes to give the high 4 bits of the message atom, with the following byte supplying the low 8 bits, and there would be a system limit of 4096 message names. If this seems low, we can use two hexades to span 8192. Either way we actually free up 1 or two hexades which can be used to encode the common stores in 1 byte as suggested by Peter.

4. In the light of (3.), we should re-examine Dave's literal statistics and consider similar immediate access to all literals, or the possibility of class-wide literal frames.

Message lookup
After a couple of excursions, I am leaning back toward essentially our current message dictionary structure, but with a decent sized cache in the interpreter for class/message/code. In this way we retain the space efficiency of having the dictionary in the class, but get fast probing from the interpreter which also bypasses the superclass lookup when successful.

The text for Smalltalk code will be carried in an optional column of the dictionary which will

> if absent, indicate a disk on which the code resides
>
> if present, contain pointers which
>
> > if absent, indicate a disk on which the code resides
> >
> > if present, point to the code

I/O Primitives
The word "primitives" is used specifically here to mean operations which are implemented in the separate I/O processor of the NoteTaker. The graphical primitives will include at least projector-oriented versions of BITBLT and line drawing, with the possibility of entire text-line operations and KAOS animation operations. In addition there must be support for display control, floppy, keyboard, touch-screen and pointing device. The 8086 also appears capable of significant musical synthesis, and we will probably support both FM and direct modes. If there is room, we will put in some speech primitives also - after all, talk is cheap (tee hee).

We have yet to establish exact timings for these operations, but they look as good or better than the current ALTO times. With more detailed information, we can choose more appropriately where to divide the load between the two processors, and therefore what the task control blocks will look like.

NoteTaker Environment
Generally, Smalltalk will run totally resident on the NoteTaker, and significant applications will use the floppy disk to read and write related data such as documents, music, source code, etc. At any time, the system can be saved in bootable form on the disk, and the time for this operation will, like booting, be around 15 seconds. When we go to 512K, there is a potential need for multiple disks to hold a single boot-file; we will throw a party for Shugart at that point.

D-0 Environment
I would like to provide a totally compatible D-0 environment - meaning that the very same boot file would run on D-0, with microcode running Smalltalk, and some Mesa runtime code handling calls to the I/O processor.