XEROX
PALO ALTO RESEARCH CENTER
Learning Research Group
May 31, 1978

From: Dan Ingalls

To: NoteTaker Smalltalk Interest

Subject: Getting Started...

File on: MAXC <INGALLS>NoteTalk3.Bravo

Now is the Time
The NoteTaker hardware schedule is very short, and writing a new Smalltalk from scratch will take a while. Many components of the Notetaker Smalltalk system can be coded at this point in time (BitBLT, byte fetch, text primitives). Of those that remain, most (ROT format, storage management, method lookup) depend on design decisions which can be argued on both sides, and which may as well be decided by fiat. In order to get us going, this memo proposes some arbitrary decisions, and outlines the tasks involved in transfering our current Smalltalk system to the NoteTaker hardware. I feel that the small machine environment will be sufficiently different that we should concentrate on getting something up, and then look at performance improvements after the dust settles. In the places where we already have good ideas, we will make sure the initial implementation admits improvement readily.

Some Arbitrary Decisions
The simplest of the ROT formats is one which uses a two-word entry for each object, in the form which the 8086 can load with its doubleword LDS (Load Data Segment) instruction. This provides high speed access to data, and makes enough room in the ROT for reference counts and garbage collector bits. It is limited to 16K objects at this level of simplicity, but that is no problem for 128K, and may even be adequate for a 256K NoteTaker.

The simplest storage reclamation scheme is a full sweep garbage collector, and, by augmenting it with an incremental sceme (see <INGALLS>CompostGC.Bravo) we should get as good or better performance than with our current reference counting. If not, then we can go back to reference counting for the incremental stuff - a possibility which must be provided for in our coding style.

Our current access to methods via MessageDicts is hard to beat for compactness and simplicity. Its efficiency can be greatly enhanced with a cache, and such an enhancement will be essential if multiple superclassing gets heavily used. I propose that we implement this scheme with cache; the latter being separately debuggable.

Given the restricted range of object pointers, we can get some extra simplicity and speed out of using 1/2 the pointer space for 15-bit integers. For LargeIntegers, we can go with Dave's current implementation for a while, later writing 8086 code to take advantage of the 8086's hardware support for extended arithmetic. Incidentally, it would be nice if we could borrow an existing 8080 or 8086 floating-point package to support Floats.

We have gotten a lot of simplicity out of our Contexts as objects, and I'm reluctant to change this scheme in the first implementation. The one thing I would change, however, is to build the tempframe into the Contexts so that allocation can go a little faster, and work out a scheme (1-bit reference count?) to recognize the common case that a returning Context can be freed.

Then What?
Assuming that we go with these or similar decisions, the NoteTaker Smalltalk environment requires implementation of the following modules:

Interpreter: byte fetch, fast ops, method lookup, activation
Storage Management: allocate, liberate, compact, full gc, inc gc
Text display: pick string, pick runs, pick font, measure/display
BitBLT: clipping, setup, actual BLT, line-drawing control

In addition, some redesign has to be done where we are going to change things:

Text display: restriction of primitives to line-level means some new Smalltalk code to handle the inter-line code.

The bootstrap writer (Ted's VMapper) must translate our bytecode change, the new ROT format, the multiple superclass hook, and the extra class field in each object and probably more...

The file system and other I/O stuff (mouse, kbd, etc) will have to be redone, meaning at least several changes to File and Directory, and probably a complete redo of the I/O part of User.

We must, at an early stage, estimate the storage requirements for the system and check them with the hardware reality; a further issue here is what number and capacity of processors we can *count on using* in every NoteTaker.

Making the Move
There is the potential in the current Smalltalk of making the bootstrapping a really enjoyable process, but I don't yet feel that I can convince anyone of how. It should be possible to build the entire NoteTaker memory image from our current system, including the 8086 code for the processor(s). It should be further possible to simulate the code, project the bitmap in a window, assemble changes to the 8086 code and all the things we would like. The only reason I can see for not doing this is the possibility that the Alto assembler, umbilical/debugger can be done sooner and will be more convenient. Even in that case, I would consider the simulated bootstrap experience to be of great interest in terms of self-definition and portability to any other environment.

Let's Talk
We need now to have a meeting to discuss these proposals, talk about individual interests, and assess manpower needs and sources (summer help?). How about this Friday at 11:00?