Self 4.1.4 Release Notes

David Ungar

1 Introduction

The Self project has not been active at Sun since 1995. Although some of its innovations (especially those concerned with implementation technology) have found their way into the world, other Self inventions such as the language design, the user-interface, the programming environment and the transporter are still not yet widely known. In order to make it possible for more people to experiment with these ideas, and for our own pleasure, I have spent my spare time porting Self to the Macintosh. Erik Ernst also helped a lot during his internship here. This system is far from polished, consider it a beta version. In particular, it includes none of the clever implementation tricks that gave Self its advanced performance on the SPARC™ processor, however it runs acceptably on a 500 Mhz G3. And, though I have used it to do many hours of work on my Powerbook (on both Mac OS 9.0.4 and OS X), it is a bit flakier than I would like. However, I am releasing it so that friendly and determined users can experiment with the system, and so that Self devotees can exploit the portability framework and VM cleanup work we have done. Following are some brief explanatory notes. Send questions to `david.ungar@sun.com`. You may also want to try sending mail to the Self interest mailing list, `self-interest@egroups.com`.

Now it is Summer, 2001, and I am putting out 4.1.4. I wish I had had more time to file off the rough edges, but believe there is enough value added since 4.1.3 to put out another release.

2 Enhancements to Self 4.0

2.1 OS X Compatibility

Since Self 4.1.2, I have ported the virtual machine and ui2 to Apple's Carbon compatibility library. This means that Self 4.1.3 will run either under Mac OS 9.0.4, or under Mac OS X. In addition, there are two variants of the virtual machine, one compiled against Carbon version 1.0.4, and the other for Carbon 1.1. Only the 1.0.4 version can be used on Mac OS 9.0.4 because 9.0.4 comes with Carbon 1.0.4. Either version can be used with the OS X Public Beta, and the 1.1 version catches exceptions so that if Self crashes, you will get a change to save a snapshot.

2.2 Portability Framework

Since Self 4.0, I have added a portability framework to the virtual machine and partially ported it to MacOS and the PowerPC architecture. The port is only partial because it only implements

enough to run Self on the Mac, not enough to realize the full performance of the Self VM. In particular, I ported the memory system, OS interface, runtime system and the non-inlining (simple) compiler (the NIC), but not the polymorphic inline caches (PICs) nor the simple inlining (smart) compiler (the SIC). I ported the frame conversion code needed to create debugging frames from unoptimized frames, but not the frame conversion code used to create optimized frames from unoptimized frames, nor the code that creates unoptimized frames from optimized frames.

With the portability framework in place, it should be easier for others to port the virtual machine. The portability framework consists of additions to `makeDeps`, a tool that manages C++ header files, and of idioms employed when writing partially- or fully- machine-dependent classes.

2.2.1 `makeDeps`

We have used `makeDeps` (formerly known as `makeDepsAndIncs`) for many years. Following suggestions by Roland Conybeare and Craig Chambers, it ensures that each C++ source file includes each header file it needs exactly once and in the right order. On platforms that use the "make" utility, this tool also creates a dependency file that informs make of the source-header file dependencies. `makeDeps` is fed a manually-maintained database ("includeDB") of interfile dependencies. It performs a transitive closure and topological sort and outputs a super-header file for each source file. The super-header file (with the suffix ".incl") is included by its source file and in turns includes the real header files in the right order.

New in Self 4.1, on platforms that support precompiled header files, this tool also selects header files to be included in the precompiled header file (based on frequency of use).

However, the major innovation for this release (thanks for Lars Bak for the original idea) is the addition of a macro facility in order to allow the file-inclusion topology to be configuration-dependent. Macros are delimited by angle-brackets and are defined in a separate platform file.

For example, in the previous release, you could invoke `makeDeps` by giving it a single file: `makeDeps includeDB` and `includeDB` might include a line such as `asm.c asm.h` meaning that the asm.c source file requires the asm.h header file.

But now, you invoke `makeDeps` with two files: `makeDeps Platform_mac includeDB`. The `includeDB` file might have a line like `asm_<arch>.c asm_<arch>.h` which would refer to the arch macro, and the Platform_mac file would include a line: "arch = ppc" defining the arch macro. Thus the result would be to declare a dependency between the asm_ppc.c file and the asm_ppc.h file. When used with the Platform_sparc file which contains "arch = sparc" the sparc assembler

files would get used by the VM. The macro facility lets us reuse the same include-file dependencies with different platforms.

Table 1 List of `makeDeps` macros

Macro name	Description	old definition	Mac Definition	Proposed PC Definition
arch	Instruction set architecture	sparc	ppc	i386
compiler	Compiler used to compile VM	gcc	mw	viscpp
os	Operating system family	unix	mac	win32
os_version	Operating system version	solaris	carbon	windows95

`makeDeps` has two more new features: `generate_platform_dependent_include` and `no_precompiled_headers` (both originally suggested by Lars Bak). I'm afraid I'll have to defer the explanation of the first of these to the section on programming conventions below. The second simply disables the use of the precompiled header file for a given source file.

2.2.2 Portability framework programming conventions

The portability framework used in this release of Self is actually a blend of two styles, an `#ifdef`-based style used where absolutely necessary and a file-inclusion-based style (thanks to Lars Bak) used where possible. The file `config.h` explains the various macros used in the ifdef predicates. For example it contains:

```
# define SPARC_ARCH 1
# define  M68K_ARCH 2
# define   PPC_ARCH 3
```

Some other agent is then expected to set TARGET_ARCH to one of these values, either a makefile (on Unix), or a file included only on that platform (config_mac.h on the Mac). #ifdefs do not scale, and the file-inclusion convention is used instead wherever possible.

The file inclusion convention has been explained in the section on `makeDeps` above, except for one case. Sometimes a class has an interface that is partly platform-independent and partly platform-dependent. When that happens, the portable part of the class is defined as usual in a file called className.h, but include an extra line as follows:

```
class snort {
             /* portable stuff goes here */
             # include "_snort_pd.h.incl"
};
```

The funny-looking `#include` includes a `makeDeps`-generated file that includes the right platform-dependent file. The includeDB file has a line:
`snort_<arch>.h generate_platform_dependent_include`
that instructs it to create a _snort_pd.h.incl file that includes the snort_ppc.h (or snort_sparc.h) file. In turn, the snort_ppc.h file contains the ppc-specific declarations for the class, which are included into the middle of the class definition. This mechanism allows a class to have nonportable attributes in its interface. It is not needed if only the implementation is nonportable because the implementation can simply go in a platform-specific source file.

2.3 UI2 on Mac

Although UI2 was originally designed for a three-button mouse, the Mac has just a one-button mouse. So, option-click on the Mac is used for a middle-button click, and command-click is used for a right-button-click. To change these, find the whichButton: method in the initialization category in the traits ui2MacEvent object.

3 Enhancements and changes to Self 4.1.4

Since the previous release:

- Several memory leaks and Macintosh performance bugs have been fixed.
- The abstract syntax module had been deleted.
- The outliner has been refactored into a pluggable outliner hierarchy and a model hierarchy. This change would facilitate supporting other languages with the Self programming environment.
- When changing a method in the debugger, if the method is present in more than one object, the debugger will ask what you want to do rather than changing it everywhere as before.
- Activation mirror objects now also contain a reference to the relevant process, simplifying their usage.
- Browsing implementors, senders, etc. is now more versatile: slice outliners have been added. These can browse hierarchically, restricting the search to an object’s ancestors, descendants, or both (its family). When used in such a fashion, the results are displayed in a visual containment hierarchy corresponding to the objects’ inheritance relationships.

- Some of the methods that were in outliner objects, but were specific to modifying Self objects have been relocated to mirror and slot objects.
- Outliners how have a “yank” menu button. This operation allows you to extract a view on just a single method or category, like “spawn” in Smalltalk-80.
- I have written a general parsing framework, and written a Java and Self parser in it. (As of this writing (8/5/2001), I’m not sure whether the parser will make it out the door.)
- A slot in the prototype module object, “comment”, has been renamed to “myComment” in order to support robust interactive editing of the comment. (The module outliner has been revamped and now supports comment display and editing.) If you have any Self source files of your own, you should use a text editor to rename the “comment” slot in the module before reading them in.
- In order to recategorize a slot in a given object, you should now “copy” the slot and drop the copy into the new category. The system will notice that the slot is merely being recategorize and not ask for confirmation.
- A cache is now maintained of those modules that are not contained in any other modules, that is the “top-level” modules.
- Several bugs have been fixed, including one that made it impossible to single-step a thread under certain circumstances.
- OS X saves moderately-sized snapshots so quickly that I have removed the garbage collection operation before a snapshot. You can always select “clean up memory” from the background menu.
- The virtual machine file name suffixes have been changed to be more compatible with modern C++ compilers.
- Some of the icons have been replaced with much nicer ones (thanks to Kristen McIntyre).

Since Self 4.1.2

- Self now runs on OS X.
- A bug in the SPARC spy has been fixed.

4 Limitations

The Self Mac port does not include any of the performance tricks that Self is known for: there is no optimizing compiler and no polymorphic inline caching. I have used a 266Mhz G3 and found the system usable if the desktop was kept uncluttered. Some operations (such as implementors of a popular selector) were pretty slow. On the newer 400Mhz G3, it feels pretty spiffy most of the time.

5 Bugs

5.1 A few really upsetting bugs are left:

Occasionally, Self crashes, freezes, or crashes the Mac. Increasing the memory allocated to the Self process seems to greatly reduce the frequency of this problem.

The SIOUX console window is limited to 64KB and will cause crashes after that. So, if, after printing out a lot of stuff, the console window goes wacky, try saving a snapshot and restarting Self.

Once in a while, when collapsing the Self UI2 window on the Mac, the UI crashes.

The outliner work has destabilized things a bit; sometimes the module dictionary keeps refilling over and over again. When this happens, I recommend that you kill the scheduler (^C fillowed by “q”), restart the prompt (“prompt start”), and force a refill of the module dictionary (“moduleDictionary alwaysRefill”).

Also thanks to the outliner work, once in a while a debugger will appear when moving a slot.

5.2 The following minor bugs exist:

The SIOUX console window pops up to the front at inconvenient times.

In UI2, when dragging a non-rectangular object (e.g. the trash can) the shadow does not conform to the object’s shape.

The Spy’s CPU bars and C-heap indicators do not seem to be quite right.

If you change a module and attempt to file it out, the transporter assumes that the application is located in a folder called `objects` and assumes that the subfolders that originally contained the source file are present.

5.3 Finally, there are several tasks yet to be done that are on my wish list:

UI1 is a user-interface that was built to show how cartoon animation techniques could enhance the user’s experience. It is partly ported to Quickdraw, but much remains to be done.

When starting a snapshot on the Mac, if the screen size has shrunk, UI2 should ensure that your windows do not open outside the bounds of the new screen.

Although you can double-click a self source file such as `all2.self` to start Self and read the file, it does not work if Self is already running.

Porting UI2 to Quartz, the native graphics layer for OS X.

Finally, the icons were sketched very quickly and could really use a cleanup.

6 Conclusion

As you have read, this release embodies a lot of work with a lot of rough edges. I hope it works for you.