Standardise Forth OOP Now

Stephen Pelc
MicroProcessor Engineering
133 Hill Lane
Southampton SO15 5AF
England
t: +44 (0)23 8631 441
f: +44 (0)23 8033 9691
e: sfp@mpeforth.com
w: www.mpeforth.com

Abstract

There are so many Forth OOP implementations that is has become impossible in practice to share Forth code that uses an OOP package. This is the opposite of standardisation and promotes neither portable programs nor portable programmers. This paper suggests a rather brutal approach to promoting one Forth OOP.

Introduction

MPE has had to deal with several OOP packages over the years. In looking at packages we have been offered recently, several have come with their own OOP packages. These are in addition to the three OOP packages supplied with MPE's hosted Forths. Accepting these packages as delivered would then require MPE to maintain a minimum of six OOP packages.

Such a solution is fraught with workload and does not encourage other programmers to write to a common standard. Rather the reverse, it encourages programmers to write yet another Forth OOP package.

Brad Rodriguez' survey [1] of 1996 included 17 packages. The situation is surely worse now. Forth OOP has been an active topic for nearly 25 years – Dick Pountain's book [2] appeared in 1987.

Solution

> *Arguing that Java is better than C++ is like arguing that grasshoppers taste better than tree bark. (Thant Tessman)*
>
> *Good judgement comes from experience, and experience comes from bad judgement. (Fred Brooks)*

Discussing existing Forth OOP is mostly an exercise in discussing deficiency, with individual authors protecting their package at the expense of the community. Since nobody is prepared to accept that another package is good enough, the only solution is to start again, abandoning all the existing packages.

All authors of existing OOP packages have to accept that, in one way or another, their packages are unacceptable to other people. We then just have to pick one package to be the basis of common practice. This is not a Forth200x exercise, it is much more a Forth library exercise.

Objectives

1) Acceptable notation
2) Documented
3) Portable
4) Debuggable
5) Champion to maintain it

Acceptable notation

People are part of the design. It's dangerous to forget that. (Anon)

The notation has to be acceptable both to users and to people who consider themselves to be experts in the Forth OOP field – this probably includes everyone who has written one. I have no strong opinions on this topic.

Documented

Documentation is like sex: when it is good, it is very, very good; and when it is bad, it is better than nothing. Dick Brandon

If the package is not documented it does not really exist in a usable form. Documentation is not just the manual, it includes the commenting in the source code. Undocumented code is just lazy code by someone who does not care about other people – it's the opposite of social programming.

There are two forms of documentation, user and implementer. Both must exist.

At the very least, in the source code every word shall have a stack comment and a description of what it does.

Portable

In order to gain traction, the package must be available on the popular commercial and Open Source Forth systems.

In order to achieve this, people have to be able to port the package. They will be able to do this if the package is well documented for the implementer as well as the user.

Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it. (Brian Kernighan)

The competent programmer is fully aware of the strictly limited size of his own skull; therefore he approaches the programming task in full humility, and among other things he avoids clever tricks like the plague. (Edsger Dijkstra)

Someone who is porting this code will make mistakes, so the code must be simple enough to be debugged. For this reason, it will have to be much better than most existing packages. Writing portable code is not easy.

The portability requirement has the perhaps unfortunate side-effect of ruling out packages that require particular implementation techniques. However, such packages tend to be difficult to port to systems without that particular implementation. It is rare to find a notation that cannot be implemented, but not uncommon to find a notation that is much easier with a particular implementation.

Debuggable

It has been discovered that C++ provides a remarkable facility for concealing the trivial details of a program - such as where its bugs are. (David Keppel)

The trouble with C++ is that it requires gurus to maintain it. Gurus don't do maintenance. (Anon)

One reason that people find OOP code hard to debug is that one of the design decisions is encapsulation – you don't know what is going on. Debugging requires you to be able override the encapsulation should you need to.

Some level of application-level debugging should be built in to the package. Some users like tools such as a “dot parser” to open up a class, for example

```
 MyLine.p1.x0 @:
```

applies the @: method to the x0 component of the line defined by points p1 and p2.

Championed

No package succeeds unless it has an interested, active and enthusiastic maintainer. There is a large number of Forth OOP packages which have no active maintainer.

Conclusions

Collaborative software development is no longer difficult. A large number of web sites exist to support software development and exposure. Standardising an OOP package is not about the Forth200x process, it is about common practice.

The hardest part of the process is to realise that common practice is more important than preserving the minutiae of a package that you are familiar with.

I have my own opinion about where I want to start from [3], but fully realise that it may not be the final design, but is just a suitable starting point.

Acknowledgements

A large number of people have informed my opinions on Forth OOP packages in recent years. Among them are
Leon Wagner, Manfred Mahlow, Doug Hoffman, Peter Knaggs

References

[1] A Survey of Object-Oriented Forths, Brad Rodriguez, 1996
http://www.bradrodriguez.com/papers/oofs.htm

[2] Dick Pountain *Object-oriented Forth*
Academic Press Limited, London, 1987

[3] FMS – Forth Meets Smalltalk, Doug Hoffman, 2010
http://soton.mpeforth.com/flag/fms/