A Survey of Object-Oriented Forths

Bradford J. Rodriguez & W. F. S. Poehlman
Applied Computersystems Group, Department of Computer Science and Systems
McMaster University, Hamilton, Ontario L8S 4K1 Canada
(This article first appeared in ACM SIGPLAN Notices vol. 31 no. 4 [April 1996])

1. Introduction

The techniques and disciplines of object-oriented programming are attractive to users of the programming language Forth. Many attempts have been made to create an object-oriented Forth ("OOF"). This article presents a brief survey of the more prominent, and more successful attempts.

Most OOFs are layered on top of Forth, and exploit the fact that Forth already possesses a mechanism for binding names to data and code (the dictionary), a mechanism for binding code to data (the CREATE..DOES> construct), separate name spaces (vocabularies), an extensible compiler, and a language interpreter. Others duplicate or recreate these mechanisms independently.

2. Features

Seventeen Object Oriented Forths have been surveyed, and are summarized in Table 1. As can be seen, not all implementors felt the need to support the commonly expected OOP capabilities. For example, encapsulation and polymorphism are not invariably linked. Some OOFs encapsulate class data and methods, but prohibit the use of the same message in different classes. Others support this polymorphism, but leave all class data and methods public.

Most OOFs allow a new class to inherit from a single superclass. In some cases, only the methods of the superclass are inherited -- thus placing a burden on the programmer to define the subclass data in a compatible fashion. Other systems allow a new class to inherit both the methods and instance variables (and sometimes class variables) from the superclass. A few allow multiple "parent" classes.

The basic Forth language has minimal support for data declarations, and some OOFs continue this attitude. But a simple extension to Forth [19] allows named data structures (like C's *struct* and Pascal's *record*), and some systems embody this for the declaration of instance variables. Others may instead (or also) allow the use of other objects as the instance variables of a class.

When a class is instantiated, the new object may be initialized either
a) by a method explicitly provided by the programmer;
b) by a method inherited from the superclass;
c) by a system default method; or
d) not at all.

The capabilities of some OOFs may be under-reported in Table 1. Since this information is taken from the literature, the absence of an "X" may simply mean that the feature was not mentioned in published reports.

TABLE 1.
Summary of OOF capabilities

*see note in individual system description

System	Early binding	Late binding	Encapsulation	Polymorphism	Single inheritance (methods only)	Single inheritance (methods &data)	Multiple inheritance	Static allocation at compile time	Static allocation at run time	Dynamic allocation	Objects as instance variables	Structured instance variables	Default initializers	Inherited initializers	Explicit initializers	Binding mechanism	Inheritance mechanism	Encapsulation mechanism	Selector representation	Syntax
[1] NEON/YERK	X	X	X	X		X		X		*	X	?		X	X	L	L	P	?	message object
[2] Pountain 1986		X	X			X		X	X		X					C	C	D	H	object \ message
[3] Pountain 1987	X	*	X		X			X			X	X	X		X	D	D	D	A	object message
[4] Click & Snow		X		X	X					X						L	L		?	?
[5] BORON	X	X		X	X											L			A	message object
[6] Hoselton		X			X			X								L			A	object message
[7] FOOPS*																L			?	?
[8] Crawford		X				X										C			?	message object
[9] CSU FORTH			X				X	X			X				X					object message
[10] YAOOP	X															T			!?	message object
[11] Bicknell		X	X		X			X				X				D		V	S	object message
[12] Maynard		X	X	X	X											C		V	!?	message object
[13] Dahm		X	X	X		?												P		object message
[14] KEVO	X	X	X							X						D			S	object.message
[15] DREAMS	X	X		X			X									*	*		A	
[16] MOPS	X	X	X	X			X	X		*	X			X	X					message object
[17] Zsoter		X	X	X		X		X		?		X			X	T	T	V	I	object message

3. Implementation Methods

Forth's milieu is frequently seen to be real-time and resource-constrained programming. While this is perhaps an unfair characterization, it is certainly true that Forth programmers are concerned with efficiency and determinism We therefore devote some attention to the implementation of these Object Oriented Forths.

Few OOFs use dynamically allocated memory for objects. Most allocate objects statically, usually at compile time. Some systems also allow objects to be statically allocated in the Forth dictionary at run time; these objects become a permanent part of the dictionary data space.

Four columns of Table 1 describe the internal mechanisms of the surveyed systems. These are identified by a single character abbreviation, as follows:

Binding.

There are four common means to bind message selectors to methods. In order of decreasing speed, they are:

Table Lookup. The addresses of the methods for a class are stored in a table; the selector indexes directly into this table.

Linear List Search. The address of the methods for a class are stored, with their corresponding selectors, in a list (usually a linked list). A linear search of this list is performed for the selector value.

CASE Statement. The methods for a class are alternative branches of a CASE statement. The routine is entered with the selector value, and the appropriate branch is taken. (Note that most Forth implementations do not implement CASE statements with a list search, so this tends to be slightly slower.)

Dictionary Search. The methods for a class are named words in the Forth dictionary. The dictionary is searched for the selector name.

Inheritance.

The technique of allowing inheritance is closely related to binding. With some simplification, the corresponding four techniques are:

Table copying. The methods table for the superclass is copied to the inheriting class.

List linking. The methods list of the class is linked to the methods list of its superclass.

Code linking. The "default" case for an unrecognized selector invokes the method routine of the superclass (or is linked to the CASE statement in the superclass).

Dictionary relinking. The Forth dictionary is relinked to search the superclass dictionary after the class dictionary is searched. Since there is no standard representation of the Forth dictionary, such systems are necessarily tailored to a specific Forth implementation.

Encapsulation.

Encapsulation of private methods and data is usually performed in one of three ways:

Vocabularies. The methods and data of the class are contained in a dedicated Forth "vocabulary." This vocabulary is added to the dictionary search order only when the corresponding class is being compiled.

Dictionary relinking. The Forth dictionary is relinked to remove methods and data from the search order. Some mechanism is usually provided to search this "hidden" dictionary fragment. Again, this technique is Forth-system-specific.

Private symbol tables. Method and data names are not stored in the Forth dictionary. Instead, private symbol tables are maintained by the OOF.

Selector representation.

The selector which identifies the desired method is encoded in one of four ways:

Index. A small integer, typically (though not always) the index into a methods table.

Hash code. An integer hash code which is used to search a list or traverse a CASE statement.

Address. An address in the Forth system. This may be the actual address of a method or a data item; or it may be an arbitrarily assigned address used in a list or a CASE statement.

String. The name of the selector is stored as a text string, which is used to search the Forth dictionary. (The other techniques may use the Forth dictionary to convert the selector name to an integer or address; in this technique the full string is stored verbatim as the selector.)

The syntax of objects and messages is given in Table 1. Each OOF, however, passes messages to objects in a different manner. When this manner is known, we describe it in the following section.

4. Specific Implementation Notes

Here we address peculiarities, special features, and noteworthy history of the systems listed in Table 1.

NEON (1984)/YERK (1992). Neon [18] was a commercial OOF, modeled after Smalltalk; when released into the public domain it was renamed Yerk. The object is not executed; it is parsed by the message word, which invokes or compiles the method. Added data structures include: class definition header, class list, method list, method stack, and object stack. Objects are statically allocated, but dynamically allocated memory can be "cast" as objects.

Pountain (1986). The object leaves a pointer to itself on an object stack; the word \ parses the message and invokes the method. Objects are unnamed (headerless) storage in the dictionary. One of the few OOFs to treat classes as objects.

Pountain (1987). Late binding is supported interpretively, but not for compiled code. The object stacks itself on an object stack and enables the method dictionary, in which the message is searched. Object indexing is at compile time; the generated code is comparable to native Forth except for the overhead of the object stack.

Click and Snow (1986). This is a SmallTalk-like OOF written in Fifth, a Forth derivative. The syntax is not described.

BORON (1987). The object binds the message, and either compiles it (early binding) or executes it (late binding). An unusual feature of BORON is a default action for objects when no message is sent.

Hoselton (1988). The object stacks itself on the data stack; the message invokes the method. Added data structures are: method list and object list. Classes are treated as objects.

FOOPS (1989). Implementation details of FOOPS, a commercial product, are not published, but it appears similar to Pountain's 1987 OOF with different keywords.

Crawford (1989). Full details are not published. The message leaves a number on the stack; the object executes the method. Message numbers are globally assigned.

CSU FORTH (1989). Binding is done by the object; the binding mechanism is not specified. Encapsulation appears to use the Forth dictionary (either through relinking or vocabularies).

YAOOP (1990). The message stores an integer; the object binds this to a method. YAOOP allows a default method when no message is sent.

Bicknell (1992). The object enables its vocabulary and places its address on the stack; the message binds and executes the method.

Maynard (1992). The message stacks an integer selector; the method is executed by the object.

Dahm (1992). This is unusual in that it is *not* an extension to standard Forth; it is a new, Forth-like language designed to be object-oriented. The object stacks its handle; the message binds and executes the method. The implementation uses an object stack and object interpreter. Classes are treated as objects.

KEVO (1992). Another new object-oriented language, related to Forth. It is a prototype-based rather than a class-based object system.

DREAMS (1994). Another prototype-based system. DREAMS achieves polymorphism through dynamic scoping (in the manner of LISP). Selected Forth words are redefined when objects are invoked, and their original definitions restored afterwards. This gives compiled code the effect of late binding, with deterministic delays.

MOPS (1995). Mops is derived from Neon. It does not support persistent dynamically allocated objects, but does allow "temporary" objects that are allocated on a stack.

Miscellaneous efforts. While not object-oriented, Kimball [20] shows an implementation of classes using Forth vocabularies, and late binding through dictionary search.

Three systems were excluded from Table 1 due to inadequate information. ForthTalk [21] is a SmallTalk-like OOF with multiple inheritance, described only in general terms. Blagoev [22] outlines a real-time OOF for a distributed manufacturing network. And Harper [23] describes an object-oriented extension to the Forth-like language RTL, which appears to support early binding through the RTL dictionary, and encapsulation through a modification of the RTL compiler/interpreter.

Some commercial Forth packages are now including object-oriented extensions; however, these are rarely described in the literature, and information is scarce.

5. Conclusion

Clearly, there is as yet no consensus toward a "standard" Object-Oriented Forth. (Indeed, there is not even agreement on whether the message should precede or follow the object.) The seventeen OOFs surveyed here have widely varying capabilities, doubtless reflecting the different needs, interests, and experience of their implementors. We hope this survey will help Forth programmers to choose the system best suited to their applications.

References

[1] Grehan, Rick. "Yerk Comes to the PC." Forth Dimensions 13, no. 5 (January- February 1992): 6-18.

[2] Pountain, Dick. "Object Oriented Extensions to Forth." Journal of Forth Application and Research 3, no. 3 (1986): 51-73.

[3] Pountain, Dick. Object-Oriented Forth. London: Academic Press, 1987.

[4] Click, Cliff and Paul Snow. "Object Oriented Programming in Fifth." Journal of Forth Application and Research 4, no. 2 (1986): 199-202.

[5] Lewis, Steven M. "BORON - Yet Another Object-Oriented Forth." Journal of Forth Application and Research 5, no. 1 (1987): 161-164.

[6] Hoselton, Rick. "Object-Oriented Forth." Forth Dimensions 10, no. 2 (July- August 1988): 15.

[7] Callahan, Jim. "FOOPS: Object Oriented Programming System." In Proceedings of the 1989 Rochester Forth Conference: Industrial Automation, June 20-24, 1989, by the Institute for Applied Forth Research. Rochester, NY: Institute for Applied Forth Research, 1989, 41-43.

[8] Crawford, Rod. "Examples of a Minimal OOP Notation For Forth." In Proceedings of the 1989 Rochester Forth Conference: Industrial Automation, June 20-24, 1989, by the Institute for Applied Forth Research. Rochester, NY: Institute for Applied Forth Research, 1989, 46-49.

[9] Abu-Mostafa, Ayman S. "CSU FORTH: An Object-Oriented Forth Implementation." In 1989 FORML Conference Proceedings, by the Forth Interest Group. San Jose, CA: Forth Interest Group, 1989, 198-203.

[10] Morgenstern, Leonard. "YAOOP: 'Yet Another' Object-Oriented Program in Forth." In 1990 FORML Conference Proceedings, by the Forth Interest Group. San Jose, CA: Forth Interest Group, 1990, 78-88.

[11] Bicknell, Roger. "Object-Oriented Forth." Forth Dimensions 13, no. 5 (January-February 1992): 23-26.

[12] Maynard, C. A. "Simple Object- Oriented Forth." Forth Dimensions 13, no. 5 (January-February 1992): 33-37.

[13] Dahm, Markus. "Object-Oriented Forth." Forth Dimensions 14, no. 1 (May- June 1992): 16-22.

[14] Taivalsaari, Antero. "Why should object-oriented Forths be based on prototypes rather than classes? The object model of Kevo." In 1992 FORML Conference Proceedings, by the Forth Interest Group. Oakland, CA: Forth Interest Group, 1992, 180-189.

[15] Brown, R. J. "Dreams." Journal of Forth Application and Research 6, no. 4 (1994): 279-318.

[16] Hore, Michael. "The Mops page." From the World Wide Web, http://www.netaxs.com/~jayfar/mops.html.

[17] Zsoter, Andras. "Object-Oriented Forth in Assembly." Forth Dimensions 16, no. 6 (March-April 1995): 11-17.

[18] Duff, Charles B., and Norman D. Iverson. "Forth Meets Smalltalk." Journal of Forth Application and Research 2, no. 3 (1984): 7-26.

[19] Bradley, Mitch. "Structured Data With Bit Fields." In Proceedings of the 1984 Rochester Forth Conference: Real-Time Systems, June 6-9, 1984, by the Institute for Applied Forth Research. Rochester, NY: Institute for Applied Forth Research, 1984, 188-192.

[20] Kimball, Vince D. "Classes in Forth." Forth Dimensions 8, no. 5 (January-February 1987): 24-27.

[21] Lindner, Stephen D. "How and Why: Multiple Inheritance Object Systems." In 1986 FORML Conference Proceedings, by the Forth Interest Group. San Jose, CA: Forth Interest Group, 1986, 263-280.

[22] Blagoev, Lyubomir. "Real-Time Object-Oriented usw-FORTH." In 1990 FORML Conference Proceedings, by the Forth Interest Group. San Jose, CA: Forth Interest Group, 1990, 332-339.

[23] Harper, Levi. "Implementing OOP in a Forth Variant." In Proceedings of the 1993 Rochester Forth Conference: Process Control, June 23-26, 1993, by the Institute for Applied Forth Research. Rochester, NY: Institute for Applied Forth Research, 1993, 58-61.