OODC: Forth OO Package for Embedded Control

Munich, November 22nd 2001

G&D: A security company

System bank note
- security paper
- bank note printing
- bank note processing
- systems and solutions

System card
- card assembly
- SW development: operating systems/applications
- consulting
- systems and solutions

Why another OO Package?
- A variety of OO packages in Forth compete for standardisation.
- Not suitable for embedded systems application.
- Lack of documentation.
- Licensing conditions denying commercial use.
- Low profile error checking and error recovery.
  There are exceptions to this.

Our Aims
- We suggest to define a language extension that allows to hide the internal mechanisms of OO packages.
- We want to promote our OO package as a powerful basis for API development.
- We propose extensions to ISO15145 so that development of OO packages gets easier.
- We propose OO for embedded systems, providing persistent and volatile fields in objects.

Overview
The presentation covers
- Classes – structure, definition, usage.
- Fields – structure, usage. Definition of field types.
- RAM Fields – usage, construction.
- Primitives.
- Exceptions – throwing and catching.
- Binding – early, late, multiple late. Visibility.
- Forward declarations, abstract classes and methods.

Features of OODC
- Static (EEPROM) and dynamic (RAM) allocation of objects.
- Some fields of static objects can be located in RAM, the content of these fields is lost (intentionally) at power off.
- User defined field types, user defined methods to work with these field types.
- A class is an object of type CLASS or one of its subtypes.
- Static methods and fields can be defined using subclasses of CLASS for creating a new class.
- The common superclass of everything is BASECLASS.
- Methods can be renamed later.
Change requests for ISO15145

We found the following problems in ISO15145:
- Interpreter and compiler cannot be "aliased", their behaviour cannot be changed. Suggestion: Defer.
- Endianness is a problem.
- Wordlists can not be concatenated.
- DOES part is never immediate.
- Deferred setting of DOES part for CREATEd data not possible.

Classes

Classes: Structure, Definition, Usage

Classes: Structures

BASECLASS describes the structure common to all objects
CLASS describes the structure common to all classes

Classes: Definition

Classes are defined in two steps.
- Interface definition
  BASECLASS CLASS PROTOTYPE: MyClass
  METHOD myMethod ( param obj −− result )
  ;PROTOTYPE
- Implementation — includes field declaration
  MyClass IMPLEMENTATION
  T_INT FIELD: Counter
  :: myMethod   APPLY Counter GET + ;;
  ENDIMPLEMENTATION

Classes: Usage

Object creation is accomplished using
MyClass OBJECT: MyObject
or
MyClass NEW

From the interpreter, only published methods can be applied to an object.
In compiling mode, there are various possible ways of applying a method to an object. The reason is an attempt to use early binding or compile-time knowledge whenever possible. Functions involved are
SUPER THIS APPLY APPLY-DO CAST CAST-STRING

Fields

Fields: Structure, Usage. Definition of Field Types.
5.11.01 Folie 13

**Presentation OODC EuroForth 2001**

**Fields: Structure**

- **FIELDBASE** describes the structure common to all field instances
- **FIELDCLASS** describes the structure common to all field defining classes

**Example:**

```
FIELDBASE
  Structure
  Offset

FIELDCLASS
  Structure
  Offset
```

5.11.01 Folie 14

**Presentation OODC EuroForth 2001**

**Fields: Usage**

Fields of the basic field types (T_INT, T_POBJ, T_PMEM) are added to a class definition as shown here:

```forth
MyClass IMPLEMENTATION
  T_INT FIELD: MyCounter
  MyClass T_POBJ FIELD: NextInstance
ENDIMPLEMENTATION
```

Field values are accessed using the GET or PUT methods of the field types via APPLY.

For T_POBJ fields the APPLY−DO function is a means of invoking a method in the object pointed to by the field.

5.11.01 Folie 15

**Presentation OODC EuroForth 2001**

**Fields: Relationships**

- **FIELDCLASS**
- **FIELDBASE**

**Example:**

```
FIELDCLASS
  Structure
  Offset: 16

FIELDBASE
  Structure
  Offset: 16
```

5.11.01 Folie 16

**Presentation OODC EuroForth 2001**

**Fields: Defining Field Types**

Basic field types: Define field type as an instance of FIELDCLASS.

Provide parameters as required by INIT method of FIELDCLASS, e.g.

```forth
1 FIELDBASE FIELDCLASS PROTOTYPE: T_UBYTE
```

The INIT, GET and PUT methods can be inherited from FIELDBASE.

GET and PUT are immediate:

```forth
:: GET
  THIS GetOffset POSTPONE Literal
  POSTPONE THIS+ POSTPONE C@ ;; immediate
```

5.11.01 Folie 17

**Presentation OODC EuroForth 2001**

**Fields: Usage**

Static objects are allocated in EEPROM. There may be reasons to store some fields of these objects in RAM.

DISTRIBUTEDCLASS, a subclass of CLASS has the provisions to create distributed objects.

Example:

```forth
RAMBASECLASS DISTRIBUTEDCLASS PROTOTYPE: myRAMClass
```

**RAM Fields**

**Usage, Construction.**
RAM Fields: Structure

EEPROM

ROM

RAM

Primitives

Target: Gain speed while allowing flexible implementation of OO systems.

**THIS** returns THIS pointer, **THIS+** return THIS pointer + offset.

**THIS-_** saves old THIS to return stack and moves TOS to THIS.

**R>THIS** restores THIS from return stack.

**THIS_X** executes a method by method id on THIS object

```
: ExecVirtualMethodById ( method_id obj −− )
  @this
  this_x
  r>this
  ;
```

Exceptions

In an OO environment, the $x_t$ of a method is not directly accessible.

CATCHING is a prefix to $CAST$, **THIS**, **APPLY**, **APPLY-DO**, **SUPER** making these words catch exceptions from the method called.

Additional primitive **THIS_C** required, providing the functionality of **THIS_X** but using **CATCH** instead of **EXECUTE**.

Throw may be misused to throw object addresses.

```
:: MyMethod ( params obj −− )
  CATCHING APPLY MyField GET
  0<> IF "exc." THEN
  
  ;
```

Binding

Binding:

Early, late, multiple late.

Visibility.
Early binding

Target: Gain speed
SUPER invokes method of super class on THIS.
CAST-VALIDATE invokes method of given class on object if object is a direct instance of that class.
APPLY executes a field accessor method at compile time, if is immediate.
Benefits: No method table lookups, no class hierarchy searches at runtime.
:: INIT (params obj -- | |
    SUPER INIT
APPLY MyField GET
::

Late binding

Necessary for polymorphism:
THIS invokes method on THIS. No class hierarchy search required, no swapping of THIS required; method id can be retrieved at compile time.
CAST invokes method of given class on THIS object if object is an instance of that class or of a child class. Method id can be retrieved at compile time.
APPLY-DO retrieves object via T_POBJ field in THIS and executes method on that object. No class hierarchy search required, method id can be retrieved at compile time. FIT method of T_POBJ fields must check type. APPLY-DO must check for NULL pointer at runtime.

Late binding Sample Code

\ search for list element with the given search key
:: List-Search (key obj -- | false |
    THIS GetKey over
    [ Key ] CAST Equals
    IF DROP THIS@
    ELSE CATCHING
    APPLY-DO NextElement List-Search
     NULLPOINTER-EXCEPTION = IF FALSE THEN
    THEN
::

Multiple late binding (public methods)

Methods of similar name can be published from disjunct classes.
The published part of the method determines the object type at runtime and searches its body for an applicable method.
ABSTRACT publishes methods before they are defined. They can be used in colon definitions then. Their content is set later on using PUBLISH.
The run time cost for using published methods is high. Reasons:
\ class hierarchy search at runtime
\ method id retrieval at runtime
\ swapping of THIS required
Note: Parameters must be compatible for published methods of same name.

Multiple Late Binding Sample Code

\ two disjunct classes publishing Method1
CLASS IMPLEMENTATION
:: Method1 (n obj -- | |
    PUBLISH Method1
ENDIMPLEMENTATION
CLASS IMPLEMENTATION
:: Method1 (n obj -- | |
    PUBLISH Method1
ENDIMPLEMENTATION
15 CLASS2 new Method1

Forward declarations

Forward declarations, abstract classes and methods.
Forward declarations

Class stubs permit definition of _T_POBJ fields before class definitions are complete. Method stubs allow compiling calls to methods before actually implementing them. If a method stub remains unchanged until the class implementation is finalised, this method becomes an abstract method, meaning the class cannot be instantiated.

Future options

- Interpretation of stack diagrams (Method signature) and optimisation/verification on basis of these.
- Extension of : to detect field accesses and compile APPLY or APPLY-DO in that case. Further extensiondetecting a sequence <class>
  -<method> and compiling APPLY or APPLY-DO in that case.
- Dynamic wordlists, allowing to load/instead classes on demand.
- Locals scheme, providing stack framing and method signatures.
- Providing enough type information to avoid type casts and type checks at run time.

Thank you very much for taking the time!

And now:

To your questions …