How OOP Relates to MWAPI

by Rodney Anderson

The M Windowing Application Programming Interface (MWAPI) is a welcome advance in the M community and makes a significant contribution to M's impact on the computer world. The M community has been waiting for a good user input-output interface standard. The object-oriented paradigm (OOP) has been talked about for a long time, and recently the understanding of OOP has increased dramatically. This article explores the relationship between OOP and MWAPI, and is the second article in the series on OOP and M Technology.

The MWAPI is a standard the MUMPS Development Committee (MDC) derived. It is a consistent approach to programming graphical user interfaces (GUIs) on many different platforms. The MWAPI enables standard M code to be truly portable across many different platforms.

The OOP applies to object-oriented (OO) languages, such as Smalltalk and C++, and allows application development to be designed and implemented using objects. Have we ever considered that the MWAPI and the OOP might be related or that the MWAPI could incorporate many OO features? If they are related, then an OO approach to programming the MWAPI can be used to full advantage. Development with MWAPI would be easier, faster, and simpler.

Why Use Windows?

Anyone who has used a computer for some time cannot fail to have noticed the worrying trend in the physical size of software packages. In the first few years of the IBM PC, an application program, such as a word-processing or spreadsheet program, would typically fit on a single 360K diskette. One diskette contained all that was needed to run the program, save data, and output on the screen or printer. Programs could even be run from floppy disk (and frequently were). Documentation appeared in a single, brief manual. Programs were only difficult to use when badly programmed.

Since those halcyon days, software packages have expanded at an alarming rate. It is now not uncommon for even the most basic of applications to be packaged on ten or twenty 1.44 mb disks. We all realize that hardware is getting smaller and cheaper, however, software appears to be getting larger and bulkier, especially the software tools to develop systems. There have been great leaps in the facilities provided by the programs, of course, and we are all capable of producing results that ten years ago were unimaginable. Even so, software producers have become so keen to outpace their rivals that most of us never use most of the features we buy in these packages.

Much of the increase in application size arises from hardware diversity; each new piece of hardware needs special instructions for the software using it. To install a new printer, as an example, the program must know the codes needed to change fonts, spacing or type style; for any display, the program must know the number of pixels on the screen and the colors available. The driver stores these special instructions. A driver is a special file supplied for each device. Therefore, the many extra disks that come with software packages are drivers for dozens (sometimes hundreds) of different hardware devices.

This is where Windows should come to the rescue. Windows takes over all communication between the application program and the hardware—the printers, screen, and serial ports. Windows includes drivers for all these devices and the application programmer can concentrate on producing code to solve a problem.

Windows gives the user a standard interface that is neat and consistent, and decreases learning time on new applications; a standard interface to the hardware, which allows one to use many platforms and reduces application size; and a standard programming environment that frees the programmer to concentrate on problem-solving.

The Main Components of M Windowing API

The MWAPI is the M Technology standard interface to GUIs. This means that the programmer can write standard code, knowing that the final code will execute on many different M vendor platforms. (There are many previous *M Computing* articles that can serve as background on MWAPI; this article only summarizes the MWAPI standard for discussion purposes.)

Windows are rectangular areas on the screen that the user sees as objects and interacts with directly by touching the window or by using the keyboard or mouse. The user interactions are reflected in changes to the graphical areas on the screen surface.

^$W is a structured system variable name (ssvn) used within MWAPI to allow access and control of windows. This window ssvn is called ^$W and has the structure ^$W (Window Name). It holds the attributes for the window objects in the structure ^$W (WindowName,Attribute) – Attribute Value.

Elements, including gadgets and menus, are also image areas on the screen with which the user interacts. The ^$W ssvn also is used within the MWAPI to allow access to and control of elements. ^$W also holds the attributes for the elements and has the following structures

```
^$W(WindowName,G,GadgetName)
^$W(WindowName,G,GadgetName,Attribute)=AttributeValue
^$W(WindowName,M,MenuName)
^$W(WindowName,M,MenuName,Attribute)=AttributeValue
^$W(WindowName,T,TimerName)
^$W(WindowName,T,TimerName,Attribute)=AttributeValue.
```

^$DI is an ssvn that stores information about the window environment. Most of the attributes of the display are read-only attributes. Some of the attributes are defaulted from the Window platform that is running.

The interaction between the user display and the application takes place through events. The user interacts with the gadgets and menus on the screen using the keyboard or mouse, e.g., pushing a button, scrolling a scroll bar, or selecting a menu option. This can result in an EVENT. An EVENT executes a DO call to a predefined label in a subroutine.

Timer elements also operate in an event-oriented manner, however, time events are not initiated by any interaction with the user.

When an event is being executed through the DO subroutine call, all event-environment information is available in the event ssvn called ^$E.

Since the MWAPI provides a standard program interface to the programmer, it has the advantages of being a simple and easy interface (since the ssvn layout is intuitive to an M programmer), and where implemented, it provides a standard interface across windowing platforms.

Basic OOP Concepts

Here is a brief summary of the concepts underlying the OO paradigm.

Each display element, data structure, or other entity in an application is an object. Objects are categorized as Classes or Instances.

M-pact

You only get one chance to make a first impression....

Contract MUMPS solutions for all your needs. Experience in a wide variety of applications:

- Clinical & Reference Laboratory
- Anatomic Pathology / Cytology
- Physician Office & Practice Management
- Nursing Home and Dialysis Solutions
- Automated Instrumentation
- Accounts Receivable / Payable
- Inventory Control
- Barcode Solutions

Value Added Resaler for technology solutions including: Scanners from PSC and Symbol, Barcode printers from Zebra and Fargo.

Technology partners with RTS, Inc. (the industries leader for remote printers and telecommunications needs).

Solutions are only a phone call away:
Call us at: (615) 834-7666 / (615) 832-4100 FAX
M-Pact is a MUMPS programming division of:
Common Cents Systems

Software Engineers

HSII, a fast-growing health-care software company in Minneapolis, is looking for Software Engineers, at all levels, for the design and development of systems using HSII's MUMPS code-generator. Minimum two years programming experience along with design and communications skills. Desired experience in one or more of the following applications: Provider Management, Membership, Premium Billing, Claims, Capitation, AP, GL, Patient Accounting *and* Patient Scheduling.

Send resume and salary requirements to:

Human Resources Assistant
Health Systems Integration, Inc.
8009 34th Avenue So., Suite 700
Bloomington, MN 55425

Composition: An application consists of many separate, independent objects. In some cases, one object (class) contains another. This is called composition and should not be confused with inheritance (see below). The first object is not derived from the second (which is a parent/child relationship); instead the first object is composed of data and the second object. These container classes are referred to as composite classes or aggregate classes.

An object encapsulates all the methods (procedures and functions) and data needed for it to operate correctly at run time. The data are usually accessible only through the public methods of the class. The value of the object's variables determines its state.

The behavior of an object is determined by its class. A class is a template for objects, each new object being an instance of a class.

New classes are derived from base classes. Each driver class can be the base class for other classes, which results in a class hierarchy.

A class inherits the methods of its base class (and all classes above it in the hierarchy). A class can add new methods or override those of the base class by redefining them.

Objects from different classes may have methods with the same names, but these methods may respond in different ways. This is polymorphism. Two or more classes that are derived from the same base class are said to be polymorphic. The word polymorphism means, literally, many forms, and in OO means that two objects may share many characteristics but also retain unique features.

Similarities and Differences

Many similarities exist between MWAPI and the OO paradigm. The first similarity is a physical one, in that the user sees the windows as objects since they are rectangular areas on the screen. The user interacts directly with these objects by touching the object, e.g., pushing a button or scrolling a scroll bar. A window object receives this user interaction through the keyboard or mouse, and changes made by this interaction result in changes to the graphical output on the screen.

The next intangible and conceptual similarity between windows and OO is the OO concept that "an object can contain other objects" or can be composed of other objects. Comparing this to the windowing standard, we see that an MWAPI window may contain gadgets and/or menus and/or timers. Therefore, the MWAPI window is an object of type window that contains other objects of type element.

An example of how we can think of objects within the MWAPI is to identify objects using the ssvn identity. For example, `^$W("fred")` might be the identifier for a window object called "fred." If we asked this object for the value of its `"TITLE"` property then we could expect a value. Similarly, `^$W("fred","G","gadget1")` might be the identifier of another object. If we asked this object for its `"POS"` property then we could also expect a valid answer. Messaging to these objects could be achieved with a simple extrinsic function such as

```
>SET x=$$^MESSAGE( ^$W("fred"), "GetProperty"
"property=TITLE")
```

One main OO concept is that "objects may have attributes or properties that describe the object." Of course, objects may have properties (or attributes) different from other objects and the values may be unique. This concept equates to an MWAPI object (window or element) that has its own private properties, e.g., `TITLE, POS, SIZE.`

One difference between MWAPI and OO is encapsulation. In OO, an object is a combination of code and data. The code is the method by which the object allows access to its data. In MWAPI, a window is an object and the callback code is the window procedure. The window attributes are the MWAPI window objects' data. In OO, the object encapsulates the code and the data. This isolates the data, only allowing access through the predefined methods/code. Encapsulation in the MWAPI environment is not enforced because the programmer can access the attributes or data of the window objects at any time without limit. Encapsulation in MWAPI can be enforced only by all programmers agreeing to a convention. This convention follows the OO world standard and would indicate that a message must be sent to obtain the value of an attribute of an object.

As mentioned, the user sees windows as objects and interacts directly with them by pushing a button or scrolling a scroll bar. These interactions are Events that send messages to objects. In MWAPI, events can be defined for actions of an object. For example, a button selection causes a SELECT event, or an UNFOCUS event occurs when a gadget loses focus. Menus and Timers also operate in an event-oriented manner. A user can choose a menu option to cause a SELECT event; timers have TIMER events.

Objects interact with each other by sending messages. An object responds to a message by finding the code corresponding to the message, taking the parameters of the message, and executing the code as a function. Windows also use messages to communicate with other windows or applications. This means that windows can call a function within the program.

This function and the parameters to this function are described by the particular message. In MWAPI, a message is usually initiated by a user event. It includes the event name and subroutine label reference defined for that event. When the event is generated, a DO command executes the subroutine label.

In OO, the code that is executed when an object receives a message is referred to as a method. A method is written in the language of the OO program. In MWAPI, methods are equivalent to the event callbacks that are executed when an MWAPI event occurs. Just like OO, which keeps a list of these methods, a list of MWAPI event/methods with associated M code and subroutine labels is maintained by the programmer. These methods can refer back to the screen or the database via object names. Very often the messages from the window inform the object of user input. For example, a push-button window object that is being pressed is an event and sends a message as a DO label routine.

Inheritance is an extremely powerful concept of OOP. As the name suggests, an object may inherit the properties of another object—its parent. Just as a child may inherit characteristics from his father, so an instance can inherit properties from its parent. Similarly, in MWAPI, a window object may inherit attributes from its parent window or from the ^$DI ssvn. For example, BCOLOR of a gadget can be inherited from the window containing the gadget. Or the COLOR of a window may be inherited from the ^$DI ssvn. The inheritance occurs only if the attribute is not defined in the original window object.

The analogy between OO and MWAPI, however, is only strong where properties can be inherited from parents. The key to inheritance in OO is that only one copy of an OO method exists, and the system locates it by searching the inheritance tree. In the MWAPI environment, however, there are multiple copies of MWAPI data structures (one for each window and its gadgets) plus multiple copies of methods (event callbacks) and therefore no inheritance tree search occurs to locate the methods.

Every method in the MWAPI needs to be separately maintained. OO allows one copy of the method to exist up until run time, so that program maintenance is simpler. Therefore, the idea that OO programming might help maintenance is worth pursuing.

Summary

The OO paradigm is related to the MWAPI. Both the OO paradigm and the MWAPI incorporate and display composition, state, encapsulation, events, messages, methods, and inheritance.

The answer to all your M computing needs

- ✓ M (MUMPS) and C Programming
- ✓ New Application Development
- ✓ Software Maintenance and Enhancement
- ✓ M Programming Classes
- ✓ Management Reporting
- ✓ Software Validation
- ✓ Experts in Hospital Information Systems

We are knowledgable in FDA regulations and federal guidelines concerning good manufacturing practices. With over 15 years experience in the M environment, we offer solutions at competative rates.

out west
CONSULTING, INC.

2450 East Speedway Blvd., Suite 5
Tucson, AZ 85719
Tel. 602.795.7076 or 800.2outwest
Fax. 602.322.0781

Since MWAPI incorporates many OO features, using OO could be a fine approach to programming the MWAPI. The next article in this series will show examples of how OOP can be used practically to program the visual presentation layers of MWAPI, Visual Basic, and CHUI. **M**

Rodney Anderson has worked with M since 1980 and has been deeply involved in graphical user interfaces, and developing MWAPI OO tools. Write him at P.O. Box 1633, Macquarie Centre, NSW 2113, Australia, or fax him at 61-2-364-8379.

Atlas Development Corporation

Atlas Development is the leading software development and consulting firm applying M-based relational database/SQL solutions. Due to recent growth, our company has immediate openings in Sales/Marketing. A new product launch with a national sales territory will offer exceptional income potential.

Candidates must have a BA/BS degree, experience with M Technology, and an entrepreneurial attitude. Any software background in a healthcare (medical or laboratory) setting is a plus. Sales/marketing experience is helpful, but not required. Please forward your resume and salary history to Atlas Development Corporation, 23161 Ventura Blvd., Suite 200, Woodland Hills, CA 91364, or fax to (818) 225-0611. EOE.