ED 422 922 IR 057 080

AUTHOR	Lorents, Alden C.
TITLE	GUI and Object Oriented Programming in COBOL.
PUB DATE	1997-00-00
NOTE	9p.; In: Proceedings of the International Academy for Information Management Annual Conference (12th, Atlanta, GA, December 12-14, 1997); see IR 057 067.
PUB TYPE	Reports - Descriptive (141) -- Speeches/Meeting Papers (150)
EDRS PRICE	MF01/PC01 Plus Postage.
DESCRIPTORS	*Computer Interfaces; *Computer Science Education; Computer Software Development; *Computer System Design; Curriculum Development; Design Preferences; Higher Education; *Programming; *Programming Languages
IDENTIFIERS	COBOL Programming Language; *Graphical User Interfaces; *Object Oriented Programming

ABSTRACT

Various schools are struggling with the introduction of Object Oriented (OO) programming concepts and GUI (graphical user interfaces) within the traditional COBOL sequence. OO programming has been introduced in some of the curricula with languages such as C++, Smalltalk, and Java. Introducing OO programming into a typical COBOL sequence presents some interesting challenges. There are a number of new concepts to introduce along with various design issues that are relatively new to OO program design, such as file maintenance and data objects. Most C++ programming courses tend to work with objects that are more user interface related, such as traditional GUI objects. This paper describes an example of using OO COBOL and Dialog Systems (GUI builder) in an advanced programming applications course. (Author/AEF)

GUI AND OBJECT ORIENTED PROGRAMMING IN COBOL

Alden C. Lorents
Northern Arizona University

Various schools are struggling with the introduction of Object Oriented (OO) programming concepts and GUI within the traditional COBOL sequence. OO programming has been introduced in some of the curriculums with languages such as C++, Smalltalk, and Java. Introducing OO programming into a typical COBOL sequence presents some interesting challenges. There are a number of new concepts to introduce along with various design issues that are relatively new to OO program design such as file maintenance and data objects. Most C++ programming courses tend to work with objects that are more user interface related such as traditional GUI objects. This paper describes an example of using OO COBOL and Dialog Systems (GUI builder) in an Advanced Programming Applications course.

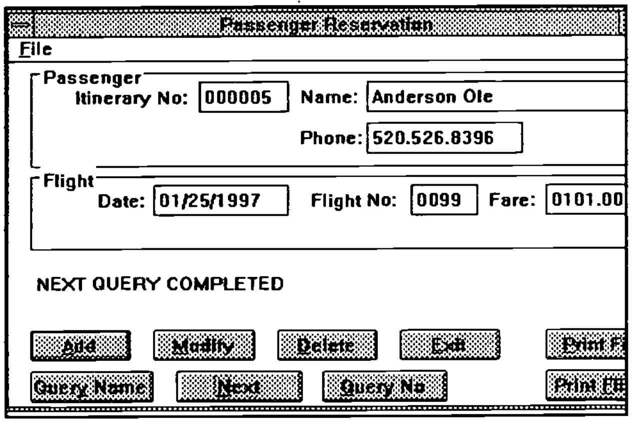
INTRODUCTION

"COBOL is in a unique position to bridge the past with the future. COBOL practitioners bring a wealth of experience spanning all phases of the software life cycle including analysis, design, implementation, database integration, and maintenance" [Arranga, 1997]. The inventory of legacy systems continues to be estimated at around 180 billion lines of COBOL code. Some organizations (typically technology and engineering based companies) have migrated away from using COBOL. They have done this by going to client-server systems using databases such as Oracle, Sybase and DB2 along with client tools such as Visual Basic, Powerbuilder, and Oracle tools. Many organizations are still committed to their COBOL based systems because of the high investment in these systems, the high cost to rebuild the systems, and the inability of the newer client-server technologies to handle the loads in large systems (salability). OO COBOL, GUI builders (such as Micro Focus Dialog Systems and IBM Visual Age for COBOL) and the integration of these products with common object standards (Common Object Request Broker Architecture - CORBA), may provide an alternative for many of these organizations to migrate their legacy systems into newer systems based on the new technologies.

GUI AND COBOL USING DIALOG SYSTEMS

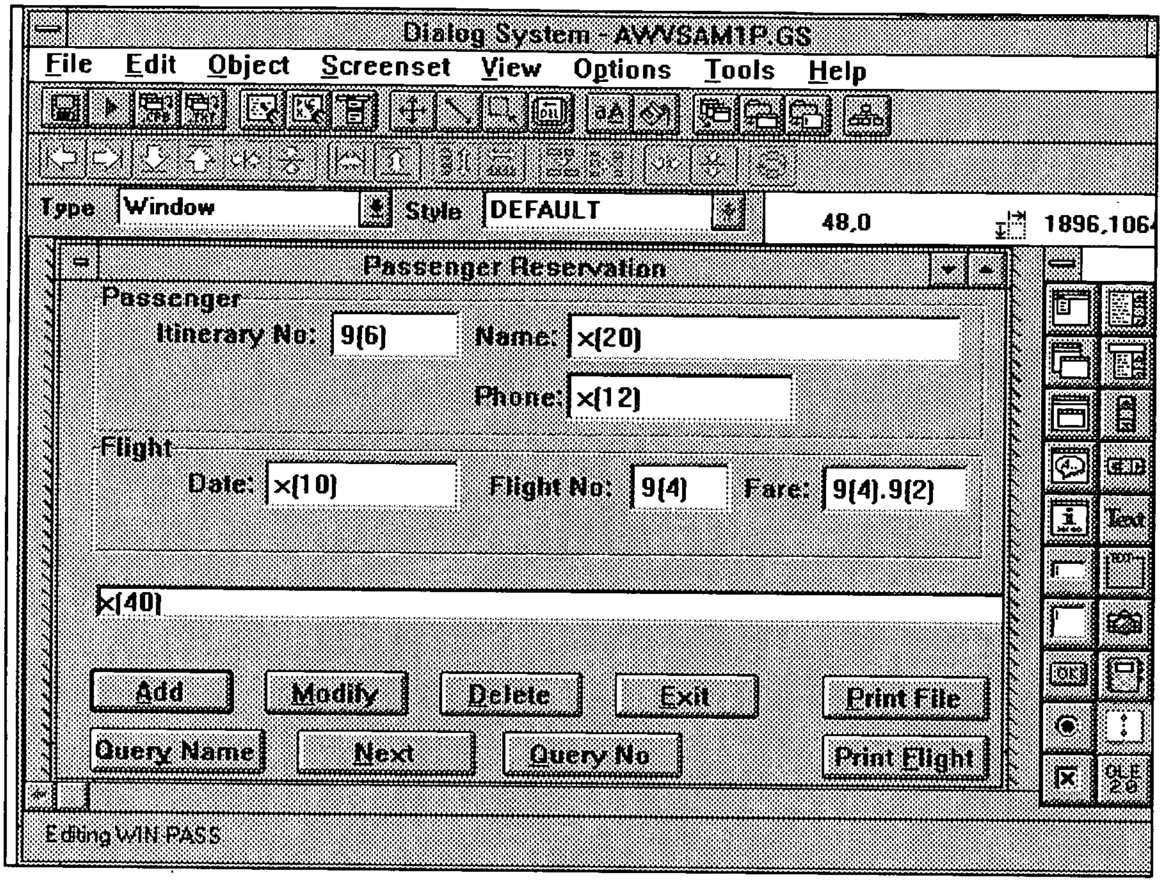
Adding GUI (windows) to COBOL program development is very easy today with the use of either Micro-Focus Dialog Systems or IBM Visual Age for COBOL. The examples shown in this paper were done using Dialog Systems. The window shown in figure 1 is a passenger reservation window that is used to maintain data using a traditional VSAM file structure with a primary key (itinerary number) and two alternate keys (passenger name and flight number).

FIGURE 1



Itinerary No: 000005 Name: Anderson Ole
Phone: 520.526.8396
Date: 01/25/1997 Flight No: 0099 Fare: 0101.00
NEXT QUERY COMPLETED

FIGURE 2


Dialog System - AWSAM1P.GS
File Edit Object Screenset View Options Tools Help
Type Window Style DEFAULT 48,0 1896,106
Passenger Reservation
Passenger
Itinerary No: 9(6) Name: x(20)
Phone: x(12)
Flight
Date: x(10) Flight No: 9(4) Fare: 9(4).9(2)
x(40)
Add Modify Delete Exit Print File
Query Name Next Query No Print Flight
Editing WIN-PASS

The window is built using the Dialog Systems object palette or object pull down menu shown in figure 2. Objects available include primary window, secondary window (clipped and unclipped), dialog box, message box, entry field, multiple line entry field, push button, radio button, check box, list box, selection box, text box, group box, bitmap and notebook.

Normally you start the build process by defining a set of data to be used with the window. The data is referred to as a data block and is defined similar to COBOL. The data block can be entered in Dialog Systems or imported from a COBOL data definition. The data block used for this window is shown in figure 3.

Normally a data element is defined for each field supported on the window along with any other fields necessary to support communication between the window and the application program. X300-Action is used to communicate a code to the application program, so the application program knows which button the user pushed when control is turned over to the application program. Each of the other fields is associated with a entry field or display field on the window. A copy block (file) of COBOL

FIGURE 3

FLD NO	FIELDNAME	FORMAT	LENGTH
1	X300-ITINERARY-NO	9	6.00
2	X300-NAME	X	20.00
3	X300-PHONE	X	12.00
4	X300-DATE	X	10.00
5	X300-FLIGHT-NO	9	4.00
6	X300-FARE	9	4.02
	X300-ACTION	X	1.00
8	X300-MESSAGE	X	40.00

definitions corresponding to each field on the window is generated by a command on the file menu. This file is called using a copy statement in the Working Storage Section of the COBOL program. This assures that the data definitions in the application program are exactly the same as the data definitions on the window.

The operation of a window is controlled by events. Events can be trapped at various levels such global (events associated with all windows in the set), for each specific window, and for each specific object on a window. Examples of events include window created, closed window, item selected, mouse over, button selected, gained focus, lost focus and various other events. Each event can be trapped at different levels and programmed to carry out various functions. This programming is called script and is executed as part of the windows operating system. Figure 4 shows some examples of script that are used on this window. Escape and Closed-Window have been defined in this application to exit the application. The application program has been programmed to exit when it sees an 'X' in the action code. The command RETC is a script command to leave the window and return to the calling (application) program. The local dialog for the window traps the exit menu selection under File and also closes the application. If the escape and closed-window events under GLOBAL were moved to LOCAL for the window win-pass, then those events would be trapped only when that window is in focus.

Script is written for each button to tell the application program which routine to execute when control is returned to the application program. When the application program returns to the window, the window program continues to execute the script that is was on when it turned control over to the application program. The ADD push button script illustrates this with the execution of the REFRESH-OBJECT $WINDOW command after it returns from the calling program. The application program sends additional data back to the window, and the window must be refreshed in order to display that data. The window is displayed at the end of any script that is executed after control has been returned to the window program.

FIGURE 4

```
GLOBAL DIALOG:
        ESC
                MOVE "X" X300-ACTION
                RETC
        CLOSED-WINDOW
                MOVE "X" X300-ACTION
                RETC

LOCAL DIALOG:
        @MNU-EXIT
                MOVE "X" X300-ACTION
                RETC

DIALOG FOR PUSH BUTTON : PB-ADD
        BUTTON-SELECTED
                MOVE "A" X300-ACTION
                RETC
                REFRESH-OBJECT $WINDOW
```

Two blocks of data (data block and control block) pass between the application program and the Dialog Systems Program (DSRUN) each time control is passed from one to the other. The application program calls the Dialog Systems program with a subprogram Call Statement. The code in the application program that is used to communicate with Dialog Systems is shown in Figure 5. The program-initialize routine is executed once at the beginning of the application to set up some of the parameters in the control block. The Call-Dialog routine is used each time the application program returns control to the window. Note that the Call statement calls Dialog-System as a subprogram using the control block and the data block.

Setting up windows like this allows faculty to illustrate all of the components of client-server programming using COBOL as the application language. The client (window program) is running various scripts that control operations at the client. The application program could be running on any platform such as an application server. If the application program has embedded SQL to an Oracle server, you are able to illustrate full 3-tier architecture in a COBOL environment. Dialog Systems supports a robust windows environment including data validation, list boxes populated by repeating group fields (tables), and multiple windows.

FIGURE 5

COBOL CODE TO SUPPORT DIALOG SYSTEMS

```
u500-Program-Initialize.
        Initialize DS-CONTROL-BLOCK, DATA-BLOCK
        Move DS-NEW-SET to DS-CONTROL
        Move VERSION-NO to DS-VERSION-NO
        Move DATA-BLOCK-VERSION-NO to DS-DATA-
                BLOCK-VERSION-NO
        Move "awvsam1p" to DS-SET-NAME
        Perform u510-Call-Dialog.

u510-Call-Dialog.
        Call Dialog-System using DS-CONTROL-BLOCK,
        DATA-BLOCK if  DS-ERROR-CODE not = 0
                Display "Dialog Error"
                Stop Run
        else
                move X300-action to 400-action
        end-if.
```

APPLICATION PROGRAMMING IN OO COBOL

The application program used in this paper to illustrate object oriented programming in COBOL is based on a model developed by Will Price in his text on Elements of Object-Oriented Programming in COBOL [Price, 1997]. I used his model to rebuild the VSAM update program that uses the window described in the preceding section. The model has a driver program, a passenger class and a database interface (DBI) class. Each class is a separate object program. The driver is a procedural COBOL program and not a object COBOL program. In this illustration, the driver contains the user interface (interface to Dialog Systems), and most of the application logic. All of the I-O has been transferred from the driver to the DBI. All of the interaction with passenger data has been placed in the passenger class. Each class is similar to a sub program in COBOL. A class can have multiple methods (sub programs) within the class. A method is called using an INVOKE command.

In object programming there are various cycles to the life of an object. The first cycle is to create the object. This is done in COBOL with the following command:

```
Invoke PassClass "New"
        Returning thePassHandle
```

The variable 'thePassHandle' is a new COBOL object reference variable that is used to maintain a pointer to the object space that is created in memory. Once the object has been created, the object can be populated with data (second stage of the life cycle) by referencing it using the handle. The sub program (method) "populate-the-pass-object" is executed using the data in 200-pass-record of the program executing the invoke.

```
Invoke thePassHandle
     "populate-the-pass-object"
     using 200-pass-record
```

The next stage in the life of an object is to manipulate or use the data. Object data can only be accessed through the methods associated with that object. The following statement would return data currently held by this passenger object to a program that wanted to pass the data to a window for displaying it.

```
Invoke thePassHandle "return-pass-data"
        returning 200-pass-record
```

The entire class program for passenger is shown in figure 6. The differences to note compared to a procedural COBOL program are as follows: 1) Class-id instead of Program-id, 2) Object Section, 3) Class control associates the class names (logical names) with physical file names on the disk, 4) OBJECT starts the definition of the object, 5) the data definitions for the object data, 6) each method definition under the procedure division is like a separate little sub program. Note that each method can have its own linkage section and local working storage section. The object data is global to all methods within this class.

A passenger object is created during the application of this program each time passenger data in read from the file, or each time a new passenger is added to the file. In this example, the same passenger handle is used for each passenger object, so once a new passenger object is created, there is no access to previous passenger objects. OO COBOL has its own garbage collection management system so the programmer does not have to mange it. More than one object of the same class can be maintained at the same time through the use of multiple handles that would be managed as part of a stack or table.

FIGURE 6

PASSENGER CLASS

```
$set ooctrl(+n)
$set sourceformat "free"
*>****************************************************************
*> Airwest Reservation System
*> 1/8/97           AWPA01CL.CBL
*>
*> This is the basic passenger class
*> Object data is:
*>   itinerary number
*>   name
*>   phone
*>   date and flight number
*>   fare
*> Methods are:
*>   return-pass-data
*>   populate-the-pass-object (invoked by DBI)
*>**************************************************

Identification Division.
    Class-id.  PassClass
           inherits from Base.

Environment Division.
    Object Section.
    Class-Control.
      PassClass is class "awpa01cl"
      Base      is class "Base"
      .
*>=================================
OBJECT.
 Data Division.
  Object-Storage Section.   *> OBJECT DATA
  01 Pass-data.
      03 200-itinerary-no       Pic 9(4) comp.
      03 200-name               pic X(20).
      03 200-phone              pic X(12).
      03 200-date               pic 9(8) comp-3.
      03 200-flight-no          pic 9(4) comp.
      03 200-fare               pic 9(6)V9(2) comp-3.

Procedure Division.
  *>--------------<*
*> Object Methods <*
*>--------------<*

*>-----------------------------
Method-id. "populate-the-pass-object".
*>-----------------------------
 Data Division.
  Linkage Section.
   01  ls-pass-data.
       03 ls-itinerary-no         Pic 9(4) comp.
       03 ls-name                 pic X(20).
       03 ls-phone                pic X(12).
       03 ls-date                 pic 9(8) comp-3.
       03 ls-flight-no            pic 9(4) comp.
       03 ls-fare                 pic 9(6)V9(2) comp-3.


  Procedure Division  Using ls-pass-data.
   Move ls-pass-data to pass-data
   .
End Method "populate-the-pass-object".
*>------------------------------


*>--------------------
Method-id. "return-pass-data".
*>--------------------
 Data Division.
  Linkage Section.
   01  ls-pass-data.
       03 ls-itinerary-no         Pic 9(4) comp.
       03 ls-name                 pic X(20).
       03 ls-phone                pic X(12).
       03 ls-date                 pic 9(8) comp-3.
       03 ls-flight-no            pic 9(4) comp.
       03 ls-fare                 pic 9(6)V9(2) comp-3.


  Procedure Division  Returning ls-pass-data.
   Move pass-data to ls-pass-data
   .
End Method "return-pass-data".
*>--------------------
END OBJECT.
END CLASS PassClass.
```

A partial listing of the database interface (DBI) class is shown in Figure 7. The database interface object is created once for the duration of the application program. The database interface has all of the file definitions and the methods to open, close, write, rewrite, read (using every index) and start (using every index or staring parameter). If this application were switched to a relational table, DBI would be changed to SQL calls to an Oracle or DB2 database. In most cases, if this change were made, the programs using the DBI would not have to be modified.

The database interface class has not defined any object data. This class defines methods only. The driver program creates the database interface object once and then invokes the object method 'open-pass-file' to open the VSAM file. From that point, the driver program can initiate read, writes, and rewrites by using the methods in this DBI.

The overall object structure of the DBI is the same as the passenger class with the exception that there is no object data. The partial listing shows one read method. Most of the read methods are all the same except that they use different indexes or start statements. Note that once the read is complete, a new passenger object is created and a method is called to populate that passenger object with the data that was just read. If the read fails the handle is set to null. The handle is used two ways. If the handle returns to the program with data in it, then the read was OK and the reference to the object is contained in the handle. If the handle is returned as a null, then the read was not OK, and the invoking program can process the exception by notifying the user.

FIGURE 7

DATABASE INTERFACE (DBI) CLASS

```
$set ooctrl(+n)
$set sourceformat "free"
*>*******************************************************
*> Airwest Reservation System
*> 1/08/97            AIRW01DA.CBL
*>
*> Airw01da maintains the VSAM file for the Airwest
*> reservation System.
*>*******************************************************

Identification Division.
    Class-id.     AirwDatabaseInterface
                  inherits from Base.

Environment Division.

INPUT-OUTPUT SECTION.

   FILE-CONTROL.
   SELECT 200-AwPass-file ASSIGN TO
              "C:\pcobwin\airwest\airw.mst"

     ORGANIZATION INDEXED
     ACCESS DYNAMIC
     RECORD KEY 200-itinerary-no
     alternate record key is 200-name
     with duplicates
     alternate record key is 200-flight-no
     WITH DUPLICATES.

 Object Section.
   Class-Control.
    AirwDatabaseInterface        is class "airw01da"
    Passclass                    is class "awpa01cl"
    Base                         is class "Base"
    .

   Data Division.

   File Section.

   FD 200-AwPass-file.
   01 200-Pass-record.
      03 200-itinerary-no       Pic 9(4) comp.
      03 200-name               pic X(20).
      03 200-phone              pic X(12).
      03 200-date               pic 9(8) comp-3.
      03 200-flight-no          pic 9(4) comp.
      03 200-fare               pic 9(6)V9(2) comp-3.

*>=================================
OBJECT.
 Data Division.
  Object-Storage Section.  *> OBJECT DATA
   01  thePassHandle       object reference.

 Procedure Division.
 *> Object Methods <*

 *>  Method Open Passenger File <*
     Method-id. "open-pass-file".
     Procedure Division.
          Open I-O 200-AwPass-file
     .
  End Method "open-pass-file".

 *>  Method Read Passenger File <*
     Method-id. "read-pass-file".

  Data Division.
    Linkage Section.
   01 ls-itinerary-no       pic 9(04) comp.
   01 ls-thePassHandle        object reference.

   Procedure Division Using ls-itinerary-no
           Returning ls-thePassHandle.

     Move ls-itinerary-no to 200-itinerary-no

     Read 200-AwPass-file
       invalid key
         Set ls-thePassHandle to null
       Not invalid key
         Invoke PassClass "New"
             Returning thePassHandle
         Set ls-thePassHandle to thePassHandle
         Invoke thePassHandle
           "populate-the-pass-object"
           using 200-pass-record
     End-Read
     .
 End Method "read-pass-file".

*> Method New Passenger Record <*
 Method-id. "new-pass-record".

 Data Division.

   Linkage Section.
   01 ls-Pass-record.
       03 ls-itinerary-no       Pic 9(4) comp.
       03 ls-name               pic X(20).
       03 ls-phone              pic X(12).
       03 ls-date               pic 9(8) comp-3.
       03 ls-flight-no          pic 9(4) comp.
       03 ls-fare               pic 9(6)V9(2) comp-3.
   01 ls-thePassHandle        object reference.

   Procedure Division Using ls-pass-record
           Returning ls-thePassHandle.

     Move ls-pass-record to 200-pass-record
     Invoke PassClass "New"
           Returning thePassHandle
     Set ls-thePassHandle to thePassHandle
     Invoke thePassHandle
          "populate-the-pass-object"
           using 200-pass-record

     .

 End Method "new-pass-record".

 Method-id. "write-pass-file".

   Procedure Division.
     Invoke thePassHandle "return-pass-data"
              Returning 200-pass-record
     Write 200-pass-record

     .
 End Method "write-pass-file".
END OBJECT.
END CLASS AirWdatabaseinterface.
```

SUMMARY

COBOL 97 and other tools such as Dialog Systems and Visual Age for COBOL are paving the way to enhance the traditional COBOL sequence in CIS curriculums into the new technologies. Much of the large corporate enterprise-wide systems world, with its 180 billion lines of COBOL code, is still looking for direction as a way to migrate. Object oriented development in COBOL may be part of the answer to this migration dilemma. IBM, Micro Focus and Hitachi have all invested large sums in the development of OO Cobol. Change in these large legacy systems over the years has always been slow. Most organizations today are too bogged down with the year 2000 problem and normal maintenance to do much experimental work with object technologies today. However, as we move past the year 2000, we will see a lot more activity with object technologies. Exposing our students to object oriented programming applications using business system applications will give them an introduction to the technologies they will encounter in working on information system development projects when they graduate.

REFERENCES

Arranga, Edmund and Frank Coyle, 'Object-Oriented COBOL: An Introduction', *Journal of Object Oriented Programming*, January, 1997.

Arranga, Edmund, and Frank P. Coyle, *Object-Oriented COBOL* SIGS Books, New York, 1996.

Price, Will, *Elements of Object-Oriented COBOL*, The COBOL Group - Object-Z Publishing, 1997.

Topper, Andrew, *Object-Oriented Development in COBOL*, McGraw-Hill, New York, 1995.

Tayler, David A., *Object-Oriented Information Systems - Planning and Implementation*, Wiley, 1992.

Obin, Raymond, *Object Orientation - An Introduction for COBOL Programmers*, Micro Focus, 1993.

Price, Will, 'Elements of OOCOBOL - Accessing Data From a Database, *The COBOL Report*, v 1 no. 5, 1997.

Lorents, Alden C., Client-Server Components and Embedded SQL using Micro Focus Workbench, Dialog Systems, and Oracle, *The COBOL Report*, v 1 no. 5, 1997.