USING INFORM, PROTOS, AND QUIZ
A User's Experiences

Richard Decker
Computer Task Group
Suite 644
World Trade Center
Baltimore Maryland 21202

This paper shares some of my experiences in developing sysems with COBOL and fourth generation languages. The fourth generation languages to be highlighted will be INFORM/3000, PROTOS and QUIZ.

INFORM/3000 is the report writing product of Hewlett-Packard's (HP) RAPID/3000. PROTOS is a product developed by PROTOS SOFTWARE COMPANY of AUSTIN, TEXAS that generates COBOL source code. QUIZ is the report writing section of POWERHOUSE products developed by COGNOS of CANADA.

A good working knowledge of the Hewlett-Packard database language IMAGE and screen formatter VPLUS must be mastered before the user can acquire all the potential that the fourth generation language PROTOS can offer. The user of QUIZ and INFORM does not need a knowledge of the database language or screen formatter. The difference is that PROTOS is a productivity tool designed to be used mostly by programmers, while QUIZ and INFORM packages are tools used by end users.

There is a learning curve to be considered when first beginning to use the new languages. I have learned all of the languages on my own without having the opportuinity of attending a formalized class on any product. Manuals supplied with all products are excellent, well written and organized. A person could become proficient in the language with about 40 hours of concentrated study and practice for PROTOS, about 20 hours of effort for QUIZ, and about 10 hours for INFORM.

OBSERVED PROBLEMS

Problem 1:

I have observed that when a fourth generation language is installed at a site a strong tendancy prevails to let the user play a predominate role in developing reports. After the database has been built, and the data loaded into the system, the screens are developed and a user can access data on-line. The user is usually very pleased with the quickness of the on-line screens, but in time, they always want a hardcopy report that shows everything. Hardcopy reports that show everything, sorted two or three different ways are not things that process very fast.

Possible solution to problem 1:

During system analysis all anticipated reports should be analyzed and an agreement reached with the user and analyst as to content before the analyst or programmer is transferred to another project. Reports that might take a long time to process might be faster in execution time if the programs creating them were written in COBOL. If the report accesses more than one file and various types of cross-checking and validation are involved within the report then the use of COBOL is recommended. Reports such as these are best addressed during development.

Problem 2:

On the other hand are 'ad-hoc' reports. These reports are those quick, short reports that answer 'questions of the moment'. Fourth generation languages are usually excellent for these types of reports. These quick reports are usually answering questions of data content. The data content is either of a certain field of data, or during a particular time period. There is usually a selection verb involved and one or more parameters associated with it. A problem arises if the programmers are busy with other things and the end user knows nothing other than the report writers of the fourth generation language. The problem is that production reports are written with the report writer, and some of the reports are made quite extensive in scope of data reported. A user can become very attached to a package and try to see just how intricate he can make a report before the entire system is brought to its knees. This type of crippling affect is not necessary.

Possible solution to problem 2:

More thought and on-going support should be given the user by the programing staff. Complicated production reports should be written in COBOL, or PROTOS, and some restraints should be placed on the user abusing the power of the report writers for 'ad-hoc' reports. This type of mal-practice causes lack of system control, and may very well cause the buying of more and more disk drives, and more powerful CPU's, all in the name of 'giving the user what they want'.

How is the programming staff supposed to accomplish this on-going support when they are already backlogged with requests. Here are a few suggestions that may help. Situations differ, of course, from site to site. The reasons for backlogs are surely not all the same.

Suggestion number one: More uniformity of programming style.

If all programs were to look alike, maintenance people would know where to look for a problem. If a change were to be made to programs, the change would be predictably placed, and a better estimate of time involved could be given. Uniformity of style also includes uniformity of code. Copylibs should be used wherever possible. Skeleton programs should be the basis for all types of programs. Screen program development, report programs, and maintenance type programs all should have a well thought out pre-defined basis for their development. The cries of programmer creativity should be drowned out. If a person has a better way of doing a routine, or program, then let the idea be shared for the benefit of all. Having more than one style of code in a shop, or worse within the same system, is a nightmare to maintain. Besides, when you execute fourth generation languages, you are trading individual creativity for speed. The speed comes from the predictable way of expressing the syntax of the development language.

Suggestion number two: More interaction between users and the DP staff.

If a user is becoming obsessed with a package, and begins to be abusive with its capabilities, do not let it continue until the entire CPU is openly crying for help. Fourth generation report writers are ideal for certain types of reports. Determination of best use is always needed.

Suggestion number three: Normalize the database whenever possible.

Fourth generation languages work best in databases that are normalized. Normalization is linking of data with the use of keys and related information. A process of delimiting the data to a smaller and smaller number of records to scan is quicker than looking at all records in a file. Random access can be used with keys. Sequential search is needed if no keys are available.

Review the present structure of the database and see if benefits could be realized with some structured modifications. This suggestion could become very expensive to incorporate on existing systems. It may not be practical or possible on vendor systems. When developing systems in-house it is a primary concern.

MAINTENANCE

Different products take different approaches to their operation. PROTOS, because it will generate COBOL code as the output utilizes its own database to organize itself. QUIZ will utilize either DICTIONARY/3000 or a compiled schema, called QSCHEMAC, to organize and operate. INFORM utilizes DICTIONARY from which it extracts data.

Schema description

A schema is a flat file describing each data element of a system in its most primary form. For example last name is a alpha-numeric element of twenty characters, and item cost is a numeric element with two decimal positions. The schema then goes on to describe where elements are found within a file. In addition to file layouts, redefinitions of a element may be found. For example the element of telephone number may be redefined as composed of an area code, prefix or exchange, and the station number or last four digits. Editing information can also be found in some schemas. This information is usually identified after a verb in the schema, such as the verb TYPE. TYPE Dollar would indicate that all elements following should be formatted with two decimal points when shown to a user. Other TYPE editing would be Date in MDY or YMD format, or a picture clause such as Type 999.9999, a number having four decimal places.

Maintenance of the schema

If the QSCHEMA method is employed then a schema file is maintained using an editor subsystem. This editor file will contain the names of IMAGE database and formfiles. In addition the names of flat and keyed files to be used are also listed. The identification of each element used, size, type, and record layout of each file is also listed in this editor file. After the editor file is created, use of a utility program creates the QSCHEMAC file.

Each data element's size and type must be entered into the system by a user at least once. Each file name and layout must be entered into the system by a user at least once. These two statements are to be done whether DICTIONARY/3000 or an editor file is used with the products. So why use DICTIONARY/3000? Because more vendor products interact with and thru DICTIONARY/3000 than an editor file.

Maintenance of the dictionary

When using QUIZ the choice of using DICTIONARY/3000 or QSCHEMA is left to the customer. DICTIONARY is probably a better choice. DICTIONARY/3000 is available on the HP-3000 whether or not a fourth generation language is in use. The dictionary could be used as a stand alone reference source. Program DICTDBM.PUB.SYS (Dict database manager) maintains the dictionary. DICTDBM is not as easy to use as the QDDR product to input information into DICTIONARY/3000. The QDDR package also has some other nice features associated with it that are helpful in the maintenance or development phases of projects. There is an option that automatically generates COBOL copybooks for all the divisions of a COBOL program. In addition a user can view all information concerning a data element, dataset, database or file residing in DICTIONARY/3000. When DICTIONARY/3000 is used there is another product made by COGNOS called QDDR that uses menus and screens to update the DICTIONARY/3000. This is important if many flat files, or KSAM files are used in the system. The record layout of flat files and KSAM files are not automatically copied into DICTIONARY/3000. Image database rootfiles (showing database schema information) and FORMSPEC, containing formatted screen information, are automatically copied into DICTIONARY/3000 using a few simple utility programs.

Maintenance of standards for programs - Copybooks

While thinking of ways to increase productivity and quality of software used in today's systems, let's not overlook some of the more common and less expensive methods available to programmers.

The communication area (COMAREA) of the buffer for the screen handler VPLUS/3000 is not simple. Use of a copybook member for the COMAREA standardizes and thus simplifies the use of VPLUS/3000. The copybook member for the IMAGE area contains all the redundant parameters the IMAGE calls require, database name field, dataset name field, modes, buffer name field, etc. Using copybooks insure that those data elements appearing in many programs will always be named the same thing. This is appreciated by those persons doing maintenance on programs after the original author has departed. It is also appreciated by the original author when trying to get a program finished by a deadline.

Another use of copybooks would be to store the format to be used within the Identification division of a COBOL program. Within this format is an expanded Remarks section. Expansion would include all pertinent information about the program and how it relates to other programs and files within the system. This can prove to be an invaluable asset during maintenance of the system.

Type of systems developed
Normally my work is in a manufacturing environment. Users of my systems include plant managers, production supervisors, operators, office clerks, and secretaries. In general, managers require summarial information, operators need simple data entry screens which require minimal training to use, and secretaries need simple and flexible capabilities with their screens. Usually the systems developed are a combination of on-line programs involving data entry or inquire screens, and batch programs. In developing the screen programs VPLUS/3000, or FORMSPEC, was the screen generator used.

Thoughts on choosing a development language
If only one of the languages were utilized to develop a system the following drawbacks might be encountered. COBOL programs take time to develop and the services of a programmer. PROTOS would require a knowledgeable programmer or user. INFORM and QUIZ are limited when it comes to doing intricate edit checking.

Factors to consider when mixing the languages
COBOL only
If COBOL is the only language used there is usually a backlog of user requests. These requests can be for new systems, modifications to systems, corrections to programs, updates to 'hard-coded' tables, or need for more programs. This delay results from a variety of factors; the quality of the programming staff, the size of the programming staff, the tenure of the staff. The DP staff's inexperience in programming, or their understanding of a system under consideration, greatly affects the speed at which requests are satisfied.

COBOL and INFORM
When COBOL and INFORM are used there is an assumption that data somehow found its way into a database, or file and a user is extracting data almost in a reference type mode. INFORM can be used to create links between various datasets in a database, but in its simplest mode it reports data elements within a single dataset. During development work this can be a great asset. The product itself is very menu/panel driven which is good for novice user, however the progression of panels gets tiring quickly. A more experienced user is given the option of stringing out his request for a panel. The more experienced user can enter a string of "1,5,3" and will get to the data element screen within the database they want, enabling them to do what they want to do. Normalization of any database is strongly suggested when using a fourth generation report writer. Normalization will affect the retrieval time for requested information. Searching for keys within the data uses a short string of data called an index to locate the desired data. If the user requests data that is not indexed then all the data must be read to match the 'string of data' requested.

COBOL and QUIZ
When COBOL and QUIZ are used a more dynamic environment can be more readily established. By using the ACCESS statment in the QUIZ language linkage to various datasets is quickly and easily accomplished. The SELECT statement is used to retrieve a subset of information. The SORT statement allows a more readable report.

Steps necessary before and during use of fourth generation languages
First the dictionary has to be created. To accomplish this run the program DICTINIT.PUB.SYS. This will establish the DICTIONARY/3000 database. After the dictionary database is created it must be filled with information. To fill the dictionary database a utility program is ran. The name of the program to fill the dictionary is DICTDBD.PUB.SYS. Filling the dictionary with information about databases and formfiles is an easy matter. With the command LOAD and the name of the database or formfile the program goes to the rootfile or formfile and gets all information of data elements, datasets, and forms. At this point the user can access the dictionary and find information concerning dataset names, and what data elements are in the datasets. To allow uers to access information located in more than one dataset at the same time, relational linkages or groups must be set up in the dictionary.

Once the groups are established a user can more fully utilize the dictionary to extract information in a more logial form.

INFORM - Run the program INFORM.PUB.SYS. A series of panels is presented to navigate the user thru the program to extract data.

QUIZ - The dictionary again has to be established prior to using QUIZ. The dictionary can be generated by using the method discussed with INFORM, or another method can be used. QUIZ is a COGNOS product, and the purchase of QDDR could also be made in additon to QUIZ. QDDR is a series of formatted screens which allow data entry into DICTIONARY/3000. Once the dictionary is made QUIZ uses its own language to extract and report data. Statements such as ACCESS, SORT, REPORT, GO, LINK and others are used to manipulate the data and datasets or files. QUIZ is powerful in that with the ACCESS and LINK statements many datasets and/or files can be logically tied together to extract needed information. No previous links need to be established, such as in using INFORM. The use of temporary subsets of information is also a useful idea. As a QUIZ jobstream executes the user can direct that subsets of information be saved either permanently or just for the duration of the jobstream. This allows for easier generation of rather intricate reports. Much more detailed reports can be generated using QUIZ than INFORM with the same amount of effort involved.

PROTOS - The dictionary involved with this product is not the DICTIONARY/3000. Rather PROTOS uses its own dictionary generated from a schema that this product recognizes. There are more steps and planning involved using this product in comparison to QUIZ or INFORM. But we are talking apples and oranges here. PROTOS is a COBOL generating language, and is more powerful in what it can do than the other choices being compared in this paper.

In planning for PROTOS a PROSCHEMA must be made. A PROSCHEMA involves naming the databases and formfiles to be accessed. Declarations of data element's type can be made. These declarations involve date formatting, currency formatting, decimal alignment and redefines of an item into sub-items. Files are declared and their associated record layouts are given. PROTOS, when defining datasets and files, will give a prefix to the items associated with them. In the COBOL code these internally generated prefixes appear. An option is available where the prefixes are predefined by the user.

Several User Defined Commands (UDC's) are then used to initialize and ready the PROTOS product for use. The three main UDS's are PROINIT, PROBUILD, and PROCOPY.

PROINIT will build the PROTOS database dictionary and initialize it for use. The dataset capacities are pre-defined, but can be overridden.

PROBUILD will read the PROSCHEMA and fill the dictionary with information as directed by the PROSCHEMA.

PROCOPY will build a copylib. In the copylib will be all the file layouts, and buffer areas that will be needed by future programs. In addition all IMAGE and VPLUS buffer areas will be in the copylib.

At this point the user has a dictionary, a linkage defining database, and a copylib. Programs written in PROTOS syntax can now be written. The PROTOS language is easy to learn and not exceptionally complicated. It is very helpful to understand how IMAGE and VPLUS work when writing a program. PROTOS is a language of few words. A program that will do quite a lot can be written with very few lines of code. This is where the increase in productivity can best be observed. However because the language is so powerful, a thorough understanding of what can be accomplished by a single command is necessary.

After the PROTOS source code is written it must be converted to COBOL code. The UDC to accomplish this is the PROWRITE udc. After the COBOL code has been generated the normal compiling,and preping must be done prior to saving the object and COBOL source code. Some installations save both the PROTOS and COBOL code, while others discard the COBOL code after the executable code has been saved. It is advisable to make any future modifications to the PROTOS source and not the COBOL source code. Within the generated code any calls to IMAGE or VPLUS are performed using the SL provided by PROTOS. Being that all error checking is done within the SL, user edit checking and exception handling must be thoughtfully done.

Factors involved when choosing a language

All three vendors have excellent reputations, and all three give very good customer support. Let us approach this question of choosing a product from what the design of the system demands.

Design Considerations

If the new system is in its budding stages the analyst must converse with the user to determine what are their expectations of the system. Will the system have many on-line screens. Will the screens be very intricate or will they display information found pretty much in a logical area of data. A logical area of data is something like basic personnel information, component makeup of a product, current payroll information, etc.

If the data elements are somehow logically linked together then those items might be contained within a single dataset. If a screen is to show many items upon a single request from the user the screen becomes more intricate. A user may like 'busy' screens that show just as much information as can possibly be displayed in eighty columns and twenty-four rows. Another user will like menu driven systems that take them to a certain area of information and display only requested items of information.

Data entry screens usually look simple enough but internally many many checks have to be done to determine proper linkage to master datasets. I am speaking of a user defined conditions that make data entry more valid and accurate. Does a purchase order have a vendor? Is the vendor valid? Cross-checking and validation of data can become as involved as the user, money, time and expertise of people involved allow.

FORMSPEC can be used to develop the screens. The programs will be written in COBOL with or without the help of PROTOS. If PROTOS is to be used there are some considerations as to how screens are made. The naming conventions are most prominent. Data element names within the PROSCHEMA will only be fifteen characters long. This limitation of field size allows PROTOS to interact with the FORMSPEC product. PROTOS allows a very convenient technique to be applied to screens working with detail sets. The technique involves filling up the screen with as many detail records as the screen will allow. The screen is defined in the PROSCHEMA using the REPEAT verb and naming of the screen. When PROTOS operates it treats the area of the screen as a matrix, and automatically tracks data fields within the matrix. With this technique inquiry, and updating of many detail records is possible with a screen full of data.

If the system is to be fairly static any of the languages will do nicely. Static means once the information is loaded into a database the user will access the system to retrieve information in much the same way as a reference library is used. The data may be updated daily, weekly or less often but at the moment the user is requesting data, the system is fairly static. Personnel, payroll, and some inventory applications are in this area of a reference system. If the system will stay within one area of a plant, then this too becomes a reference type system showing a status of the area.

When systems cross departments, or location boundaries the systems become more interactive and more complicated. More cross-checking and validation is needed to insure accuracy and smoothness of operation between the various areas of the system. In these situations a language other than a report writer is needed. COBOL programs, or programs generated by using PROTOS, are needed to build the system. If many reports are needed PROTOS is very well suited to generate complex reports with minimal effort on the part of the programmer. Reports written in PROTOS are much easier to write and modify than a straight COBOL program. The reason for this is the use of the FORMAT verb in PROTOS. The program shows the 'format' or report layout of the report in the program. The fields are then defined by just naming the data elements to be shown, and control breaks are identified by naming data elements after the BREAK verb.

QUIZ is a very powerful report writer. It is easier to use than PROTOS. If the report is a listing of data element values it is very easy to use. The user accesses the dataset or file, and gives the REPORT verb naming the data elements desired. Then the GO verb is used and the report is formatted, column headers included, and generated to printer or other device. If subsets of information are needed, the SELECT verb makes quick work of extracting the needed data. A SORT verb is also easy to use and declare. Again, if the report is a set or subset of information QUIZ is excellent to use. The way that the database is organized can greatly affect the performance, but that is the problem of the analyst or database designer, not the language. It is when the data to be extracted is not a simple subset of information that QUIZ is more difficult to use. Extraction becomes difficult if the conditions to select data are based on data elements not defined within the dataset itself.

Summary

When using fourth generation languages organization of data and preparation for use of the tool is very important. Various types of dictionaries and various dictionary utilities have been reviewed in this paper. Data structure has been shown to be the foundation for the fourth generation languages. Some dictionary maintenance products have been mentioned and briefly reviewed. In reviewing these products emphasis has been placed on how the product is set-up for operation. In general, the complexity of a program will dictate what type of fourth generation product to use for an application. The more complex a program the greater the need for COBOL. Once the dictionary is built and loaded with data of the system, extraction of information has been greatly simplified by these languages. It is this simplification of extraction that can affect system performance. Reports should be scrutinized to determine if the power of the languages is being used effectively. System administrators should be aware of how a user is accessing the data to determine efficiency and "bottle-necks" caused by abuse.