STRUCTURE OF A FILE ORIENTED PROGRAMMING LANGUAGE-GPLAN/BL/I

Robert H. Bonczek and Andrew B. Whinston

December 1975 WP-75-164

2361
Laxenburg
Austria

International Institute for Applied Systems Analysis

Structure of a File Oriented Programming Language-GPLAN/BL/I

Robert H. Bonczek Andrew B. Whinston

1. Introduction

Modern computer science has developed languages along many distinct paths; three are: Operating System Languages (e.g., OS/JCL), High Level Procedural Languages (e.g., FORTRAN), and High Level Non-Procedural Data Base Languages (e.g., ALPHA, SQUARE, GPLAN). The purpose of each of these is to solve a particular problem, namely, to simplify the work of the programmer, so that a majority of his time could be spent on his own application. The development of data manipulation procedures has also proceeded independently; of programming languages only in the programming language LISP are data and program expressed in a common manner.

We feel that the time has come for a new approach to computer language evolution, especially for business oriented users. The combination of data, programs, and operating system into a single language would make a great simplification of the current state of affairs. The BL/I language is presented as a prototype for a data base oriented computer system, combining features from operating systems, programming languages, and data base languages.

1.1 Operating System

Most operating systems include a job control language: a procedural language for specifying job steps. This includes device manipulation, large file manipulation, compilation and execution. The transparent part of the operating system is the software for scheduling, queueing, and device operation. In the proposed system, the file manipulation capability of a job control language is extended to handle arbitrary sections of a data base. Other features would remain the same.

1.2 Programming Languages

Languages such as FORTRAN and COBOL suffer from the defect of being record oriented languages, i.e., processing only a single record at a time. BL/I will also handle the record oriented programs, but it contains a set of D.B. oriented manipulation routines, where entire sections of the D.B. can be addressed from within a procedural language program. Since FORTRAN and COBOL are subsets of BL/I, it is clear that an increase in computing power is obtained.

1.3 Data Base Languages

Two kinds of data base languages have been developed. Record oriented and Procedural Data Manipulation Languages (e.g., GPLAN/DMS) were proposed by the CODASYL D.B.T.G. These usually consist of a set of COBOL or FORTRAN callable subroutines, which maintain and manipulate record occurrances in the D.B. File oriented nonprocedural languages (e.g., ALPHA, GPLAN/QUERY LANGUAGE) use data descriptions to generate output files from the data base. However, the nonprocedural approach is limited by the difficulty of comprehending complex statement and commands. Natural language processing is (as of now) too time consuming for use in this application.

2. Features of BL/I

As stated above, BL/I will attempt to combine many of the features of operating systems, programming languages, and data base retrieval into a single language. In this section are listed some of the user oriented features that BL/I should contain.

2.1 Data Base

The fundamental underlying concept of BL/I is that it is built upon a CODASYL type network data base. The entire BL/I system is described by a network structure, in that all parts of BL/I relate to others in both explicit and implicit ways. The implementation of the data portion of the D.B. is described in the GPLAN DMS Users Manual [1]. A useful feature of the GPLAN system is the ability to maintain data occurrances in LIFO and FIFO lists

(or stacks and queues), as well as sorted by key. This device is useful for the implementation in the operating system of a scheduling process.

2.2 Data Base Manipulation

The foundation for all of BL/I's DB Manipulation capabilities is the GPLAN Query Language [2]. This is a nonprocedural high level query language that can be used to compute simple and complex expressions on data, as well as to perform conditional retrieval on the data stored in the D.B. The prototype of a GPLAN is

<COMMAND> <ARGUMENTS> <CONDITIONS>

such as in: LIST STUDENTS FOR TEACHER = 'SMITH'.
However, by using a transformational grammar to parse the query, the parts of the query can be convoluted, as in

FOR TEACHER = 'SMITH' LIST THE STUDENTS

For a fuller description, see [2].

The QL has the ability to modify the data base, as well as perform retrieval. New and/or temporary record types can be created/deleted by using the proper statements. The value of having the QL as an integral part of the BL/I language is twofold: (1) it provides quick data retrieval capabilities for nontechnical users of the D.B. By presenting as english-like QL, the system can be utilized by anyone requiring it. (2) it frees the technical user from the chores of data manipulation, so that more of his time can be spent on solving the problems of his own application.

2.3 Record Manipulation

Some users of data information systems have need to actually examine and retrieve data internally on a record by record basis. To satisfy these users, BL/I includes the GPLAN Data Manipulation Language. This set of routines perform all of the necessary data base accessing functions for a program. Callable from any source language, they provide another method for modifying and retrieving

information, so that the user only need be concerned with his own computations.

What we actually have is a hierarchy of methods of accessing the data base. At the most primitive level, a user can be concerned with paging, pointers, etc. This chore is performed for the user by the DML. The user of DML is then only concerned with logical relationships among data elements, explicitly defining and computing these. This task is performed by the query system, leaving to the user the job of naming the sets of data he desires. Finally, BL/I provides a means of working with these sets of data, so that the user only need specify what processes he wants performed, and not how to perform them.

2.4 Input/Output

The input/output functions of BL/I are performed by the GPLAN/GENERALIZED LOAD PROGRAM (GLOP). GLOP has the capability of transferring data from any kind of network structure into any other kind of network structure. Note that this includes the special case of transferring a sequential file (single record type) into a network DB structure.

For example, suppose that the structure shown in Fig. 1a (assumed to be input) is to be mapped into that of Fig. 1b. The necessary structural information would be to :

Teacher Name # of Students
  Student Name Grade
  ⋮
Teacher Name # of Students
  Student Name Grade

FIG. 1a

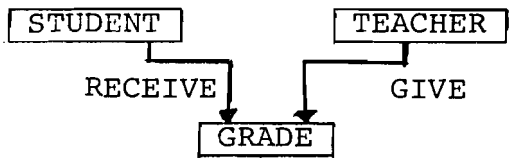

FIG. 1b

CREATE RECORD TYPES STUDENT, GRADE, TEACHER, with relation RECEIVE holding between STUDENT and GRADE and relation GIVE holding between TEACHER and GRADE. The necessary GLOP file description is:

```
FORMAT 1     (          )
TEACHER,NAME:  OWNER OF GIVE
DETAIL:  2
FORMAT 2     (          )
STUDENT,NAME:  OWNER OF RECEIVE
GRADE:  MEMBER OF GIVE AND RECEIVE
```

The term DETAIL refers to the number of records of Format Type 2 that occur. Note that only specific structural information need be given to GLOP; it is a high level language, in that only what to do is specified, and not how to do it.

The GLOP structure contains formatting capabilities. Thus GLOP can be used as a report generator for output (the GPLAN QL has a similar capability). The same holds true for input. The important idea here, though, is that conceptually input and output are just two pre-defined record types of the system. In fact, all external devices on the computer can be (logically) considered record types of the D.B. Thus GLOP can be used to manipulate all input and output. Moreover, it is possible to structure the record types for the mass storage devices, and so logically obtain storage hierarchies.

2.5 Application Program

BL/I allows both FORTRAN and COBOL programs to be compiled and executed as part of the BL/I system. In order to do this, both compilers and a loader must be BL/I resident. Logically this can be accomplished by defining COMPILER and LOADER record types, and storing the appropriate programs by name as record occurrances. In this way, many different compilers can be accomodated by the system, as well as several different versions of loaders. All of the program compiled by BL/I would have access to both the DML routines and the GLOP for processing purposes.

Along these same lines, it is possible to store user application programs in a record type, either in source or binary version. Having the program source code is useful, since

modifications can be made to it from the GPLAN QL. These stored programs can be executed from BL/I by name.

2.6 Planning

BL/I is designed primarily for a manager who is interested in getting results from the computer rather than programming it. Thus it is useful to include in BL/I a planning mechanism, like STRIPS [3], so that the manager can be aided in making whatever decisions must be made. But an economic planning module must be able to do more than a robotic planning module, because the former must often choose between alternatives according to some criteria, e.g., maximizing profits by investment.

The only way such a mechanism can be realized is to have it operate interactively with the manager. In this way, the system can query the manager concerning the decisions that must be made, in order to use his judgement on reducing the possible choices. Only with a guided interactive search can this form of planning mechanism operate.

2.7 Security

A security system has already been designed for the GPLAN Query System [4]. In this system, both operators and data values can be locked out for a particular user. This system could be extended to BL/I application programs and system routines in a simple and straightforward manner. Thus, no operation not allowed for a user could be performed by that user, nor could data values be retrieved to which that user was not entitled.

3. BL/I Control

The BL/I language uses a general recursive control structure that is quite flexible for meeting the needs of users. Since all of BL/I is defined with respect to the universal data base, the control structure is also so defined.

3.1 Storage of Semantic Information

Consider the context free grammar $G = (N,\Sigma,P,S)$, where $N = \{S,T,O\}$, $\Sigma = \{(\ ,\ ),\bar{},+,\times,\div,a_1,a_2,a_3,\ldots,a_k\}$, and P contains

productions $S \to SOS$, $S \to \bar{S}$, $S \to (S)$, $S \to T$, $O \to +$, $O \to \times$, $O \to *$, $O \to \div$, $T \to a_1$, $T \to a_2, \ldots, T \to a_k$. This grammar is called a picture grammar if we let the terminals a_i, $i = 1,\ldots,k$ be arrows ($\to$), and define the operators by

a+b	means	$\overset{a}{\to}\ \overset{b}{\to}$	hd(a+b) = hd(b), tl(a+b) = tl(a), hd(a) = tl(b)
a×b	means	$a \curlyvee b$	hd(a×b) = hd(b), tl(a+b) = tl(a) = tl(b)
a*b	means	$a \cap b$	hd(a*b) = hd(a) = hd(b), tl(a*b) = tl(a) = tl(b)
a÷b	means	$a \curlywedge b$	hd(a÷b) = hd(a) = hd(b), tl(a÷b) = tl(a)

where hd(x) = head of arrow x, tl(x) = tail of arrow x. $\bar{}$ (overscore) is a special operator; if $\bar{x}$ appears in a string, it means that this occurance of x is the same as the previous occurance of x, if one exists.

Using this picture grammar, a network data base structure can be described as a string in the language of the picture grammar. For example, the network pictured in Fig. 2 can be described by

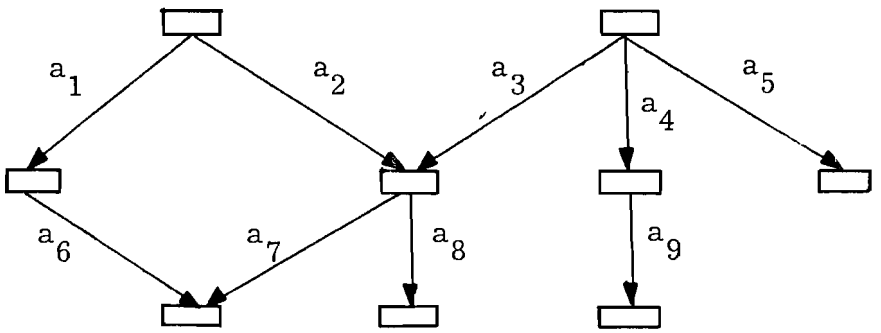

FIGURE 2

$$a_5 \times [(a_4+a_9) \times (a_3 \div [(a_1+[a_6 \div (a_2+[a_8 \times a_7])]) \times \bar{a}_2])]$$

Another description is

$$(a_1+a_6)*(a_2 \div [a_5 \times ([a_4+a_9] \times a_3)] + [a_8 \times a_7])$$

A problem, then, is that for each data base structure there exist more than one picture language string describing that structure. Any algorithm described for translating networks

into picture language strings then will be dependent upon the order the links of the networks are processed in. For a given network, the problem of finding a best representation of the network by a string is yet to be solved.

However, each well defined string in the picture language describes one and only one network structure. Thus, given a string, it is possible to completely determine the network data base the string describes. Moreover, the semantic information about the links (relations), which define the data base, can easily be stored with the string in a network structure.(Fig. 3).

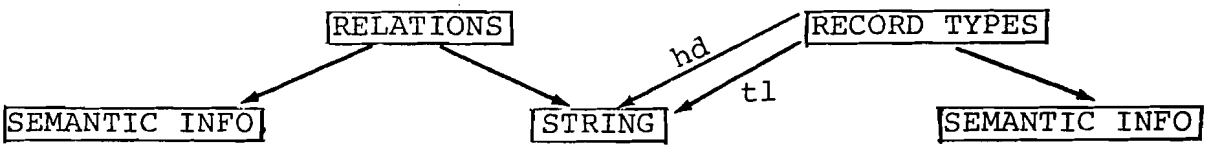

FIGURE 3

Each terminal symbol of the string represents an occurance of the record type string. Each link in the string is owned by a relation, as well as two record types (head and tail). The semantic information stores corresponds to information now stored in the record table and set table of the GPLAN Data Manipulation Language [1].

The structure allows full flexibility to be achieved in the storage of the data base schema information. One can access this information by relation name, record type name, partial semantic data, or by part of the defining string itself. This latter capability is most useful when a restructuring of the D.B.structure is required; by specifying changes in the structure of the string, the structure of the D.B. is modified.

3.2 General Description of Routines

Consider the universal data base, with record types for all programs and devices as well as data. Suppose a program

stored as an occurrance of record type A is to be run, and this program may need results from two other programs of type B and C (see Fig. 4).

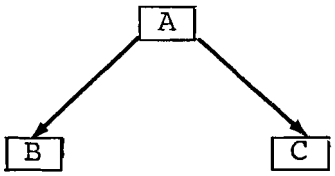

FIGURE 4

The execution of B and C is dependent upon the arguments supplied to A. The control routine for program A might look like this (in a loose LISP framework)

```
(EXECUTE X (COND ((C1) (AND (REQUEST B) (REQUEST C)) )
                 ((C2) ((REQUEST B)))
                 (T    T) ) )
```

where if C_1 is satisfied, both B and C must be run; if C_2 is satisfied, then only B must be run; otherwise neither need be run, so the value of EXECUTE would be T. The request function might look like:

```
(REQUEST X (COND ((C1) (EXECUTE X))
                 (T      NIL ) ) )
```

where if C_1 is true then X should be run; if C_2 is true then X need not be run, i.e., X halts with no output; otherwise X cannot be run, and hence the request has failed. Clearly, when EXECUTE X is issued, it may need to recursively request other routines, etc.

In general, for each node (record type) in the control structure network, there is an associated list of "EXECUTE"

conditions, which form the COND clause of the REQUEST function. Similarly, for each line (relation) there is a corresponding list of "REQUEST" conditions, which form the COND clause of the REQUEST function. Then to perform any step in the control process, all required steps must be executed (with a value of T).

3.3 Examples of the Use of the Control Process

The interface of application programs with the data portions of the DB can be facilitated by means of the Control Structure (Fig. 5).

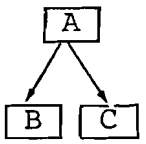

FIGURE 5

The execution of program B from program A can be effected by the process described above. Suppose program A needs some data from the D.B. in order to run. Record C would then only need contain (1) a call to GLOP and (2) the description of the desired data and the form of the retrieved data. If the data is unavailable, the request would fail.

It is occasionally useful to store a program as part of a data record occurance; this is feasible when the program is smaller than the output generated. The program itself must then have an associated "EXECUTE" list. In performing retrieval on this data occurance, the execute list could have in it a reference to EXECUTE the stored program. In conjunction with the previous example of program interfacing, this technique can become quite powerful.

Complex BL/I programs can be constructed using this technique. By providing a user defined control structure capability, a programmer can build an EXECUTE-REQUEST network of BL/I subprograms. Moreover, if BL/I is operating in a multi-processor environment, then the control structure can be used to specify routines which can be processed in parallel.

3.4 BL/I Control Monitor

The actual BL/I Control Program is designed according to the BL/I Control Structure. For each kind of BL/I instruction, there is a record occurrance, with its associated "EXECUTE" list of conditions. Each record occurrance is then linked to appropriate other occurrances. In this way, security is a fairly reasonable thing to implement: if the user does not have the proper security clearance, then his request will fail (have value NIL).

Of course, the BL/I Monitor is also an operating system, so that it must also contain routines for resource management and scheduling. The Control Structure is a logical description of the method of program execution. The interface of the application side of BL/I with the operating system side must be made consistent.

The actual form of the Monitor has yet to be established. To proceed in this direction, the external routines of BL/I must be completely specified. Once this is done, we can proceed to implement the internal structure of the language, using the criteria specified by the requirements of the routines. This is the next step in the development of the BL/I language.

However, it is clear at this time that, besides being a language oriented toward management applications, it also has many of the properties of LISP that are desirable. The D.B. capability is at least as powerful as data structures definable in LISP; by using the QUERY LANGUAGE to do retrieval, the burden of using CAR and CDR is removed. The control language is recursive. Further, a LISP compiler or interpreter could be included as part of the compiler record type, so that existing LISP routines could be incorporated into the system. With the Planning System, BL/I becomes a useful tool for researchers in artificial intelligence.

4. Conclusion

This paper is an outline for a new, high level procedural computer language that would encompass ideas from operating systems, programming languages, and data base technology. The language would incorporate many features of these three areas, with more interrelations among them than is commonly found today. However, much work still needs to be done on this subject. In particular, the control structure for BL/I needs to be worked out in greater detail. Furthermore, the economic planning mechanism must be designated to easily interface with the data base. We believe that the realization of BL/I will be a great step in the evolution of management oriented computer languages.

References

[1] Bonczek, R.H., Cash, J.I., Haseman, W.D., Holsapple, C.W., Whinston, A.B., Generalized Planning System/Data Management System(GPLAN/DMS),Users Manual, Krannert Graduate School of Industrial Administration, Aug. 1975.

[2] Bonczek, R.H., Haseman, W.D., Whinston, A.B. "Structure of a Query Language for a Network Data Base," Working Paper.

[3] Fikes, R.E., Nilsson, N.J. "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," Artificial Intelligence 2 (1971), 189-208.

[4] Cash, H.I., Haseman, W.D., and Whinston, A.B. "Security For the GPLAN System," submitted to Information Systems, Feb. 1975.