Non-procedural data processing

B. M. Leavenworth

IBM Thomas J. Watson Research Center, PO Box 218, Yorktown Heights, New York 10598, USA

The specification of data processing applications in a nonprocedural manner is characterised in terms of the following features: elimination of arbitrary sequencing, pattern directed structures, aggregate operations and associative referencing. The paper defines a simple data processing problem, then motivates each of the features with respect to different descriptive aspects of the application. It is shown how these features form the nucleus of a nonprocedural language called the Business Definition Language to state and solve data processing problems.

(Received May 1975)

Introduction

By nonprocedural data processing we have in mind some of the following characteristics:

1. The specification of data processing applications in terms of only those abstractions which are relevant to the problem.
2. The specification of the outcome desired as a function of the inputs.
3. The suppression of unnecessary detail.

One of the earliest attempts to address the problem of nonprocedural specification in general was the information algebra (Codasyl, 1962). Although more of a notation than a language, the information algebra had many of the characteristics to be discussed here. In Leavenworth and Sammet (1974), certain features were proposed which were believed to characterise some of the attributes of nonprocedural languages. We include and discuss here all those features with the exception of 'nondeterminism and parallelism' which is subsumed in the present context by 'elimination of arbitrary sequencing'. The features are:

Elimination of arbitrary sequencing
The elimination of any sequencing not dictated by the data dependencies of the application. A more satisfactory term might be 'lack of sequencing constraints'.

Pattern directed structures
The execution of a program based on a pattern or template rather than an explicit sequence of commands.

Aggregate operations
The application of operators to entire aggregates considered as single entities.

Associative referencing
The accessing of data based on some intrinsic property of the data. This facility is usually found in languages that contain sets as data structures.

The present paper first defines a simple data processing problem. It then motivates each of the features with respect to describing aspects of the problem and shows how they can form the nucleus of a language called the Business Definition Language defined by Hammer, Howe, Kruskal and Wladawsky (1974; 1970) to state and solve data processing problems.

A data processing application

The sales analysis application to be described was given by McCraken and Garbassi (1971) as representative of a data processing problem to be programmed in COBOL. The outputs desired are two reports of products sold by a company on a regular basis. The reporting period could be over a period of a month, a week, or some other interval of time. The reports are the following.

Report-1, summary by product
For each product, list the total sales for that product and the year to date sales.

Report-2, summary by district and salesman
For each district, list the salesmen in the district giving the total sales for each and list the total sales for the district. List the total sales for all districts.

To produce the reports, the inputs used are the transactions realised during the reporting period, and a master file. Each transaction document (called Detail here) contains the product number, quantity sold, and the salesman and district identification. Each record in the master file (called Master) contains the product number, the unit price, and year to date sales.

The formats of the inputs (Master's, Detail's), outputs (Report-1, Report-2), and an intermediate collection of records called Ext detail's are shown in **Fig. 1.** The basic flow of the required processing is shown in **Fig. 2.**

Elimination of arbitrary sequencing—data flow

There are two aspects of sequencing to be discussed. One has to do with the flow of data between the organisational entities or steps (see below) of an application. The other relates to the data dependencies with respect to a particular step. We treat the first aspect in this section and discuss the second later in the paper.

Arbitrary sequencing is eliminated by representing an application by a data flow network (Hammer, Howe, Wladawsky, 1974; and Kruskal, 1970). By decomposing the application into a set of steps which communicate with one another only across linking paths, the sequencing is governed strictly by data dependencies, i.e. one step cannot consume data until it has been produced by its predecessor steps.

In Fig. 2., the application is represented by a directed graph where the nodes are called *steps* and the arcs are called *paths* over which collections of documents or records flow. *Files* may be connected both to the inputs and outputs of steps. There is one file called Master's (the apostrophe followed by 's' is used to denote a collection or group) in this application. There are three types of steps represented here: Source steps, Sink steps and Transformation steps (Tran-1 and Tran-2). Tran-1 has two inputs: detail records (Detail's) from the Source step and the file Master's. It produces three outputs: Report-1, which is consumed by Sink-1, Ext detail's which are input to step Tran-2, and Master's. It is convenient to represent steps with multiple outputs. However, since each output defines a separate transformation, it is described by a separate program.

The programs to produce Report-1 and Report-2 (**Fig. 4** and

Downloaded from https://academic.oup.com/comjnl/article/20/1/6/340970 by guest on 19 April 2024

Master	Product number Unit price Ytd sales
Detail	Product number Quantity Salesman District
Ext detail	Ext price Salesman District
Report-1	Product number Product total Ytd sales
Report-2	Salesman Salesman total District District total Month total

Fig. 1

Fig. 5 respectively) are written in the Business Definition Language, as well as a program to produce an intermediate result (**Fig. 3**).

A step with multiple outputs can be considered to be a multiple-valued function, since the paths can be executed in any order or simultaneously. The only requirement is that the step have all its inputs before it can execute.

The advantages of a data flow model is that the execution is determined only by data dependencies; a step contains no references to other steps in contrast to conventional programming protocols.

Pattern directed structures

The execution of a step is driven by a pattern or template which describes a prototypical element of the group produced by the step. Fig. 3 shows the specification for the production of Ext detail's as a function of Master's and Detail's. The complete meaning of this description will be gradually explained in this and subsequent sections. An important characteristic of pattern directed structures is the use of prototypical elements to specify transformations implicitly rather than explicitly.

Each program or pattern has two parts: a specification and a definition. There are two columns associated with each part. For the specification, the first column contains either the name of a group of documents, or the name of a field in the document being defined. The second column specifies the value of the field or the cause of a prototypical member of the group. For example, the first line of the pattern in Fig. 3 specifies that each Ext detail is produced from one Detail. In most cases, each name used in the specification of a line of the document is local, i.e. has meaning only for that particular line. Also, each name used in the specification part must be defined in the definition part. By a line, we mean a logical line which contains all the information needed to complete a specification or definition; it may require several physical lines. For example, the line referring to the specification of District in terms of Old District includes the corresponding definition of Old District and Detail; the logical line in this case consists of two physical lines. A naming rule is introduced here which allows one to use adjectives (Old in this example) to distinguish between the field name and the name of its value. Note that the use of level numbers follows COBOL usage to show the components of a structure.

For the definition part, the first column gives the name being defined while the second column gives the definition of that name. For example, Detail's is defined as one of the input groups to the step being described.

An exception to the locality of reference rule is shown as part of the definition of Salesman. The name Detail is defined as the CAUSE OF Ext detail. Detail is global and refers to the document that is responsible for producing the current Ext detail. Further clarification on the use of names is given in the next section.

It is generally appreciated that a program is a static description of a dynamic process (the computations that take place at execution time). For example, a local reference in a recursive procedure or function successively represents the different incarnations of the variable at execution time. There is a similar duality between the declaration of a document as a pattern and the program which produces instances of that document. One of the aims of nonprocedural specification is to represent the dynamic process by a static description which is relatively simple, functional and deals with the abstractions which are meaningful to the problem definer.

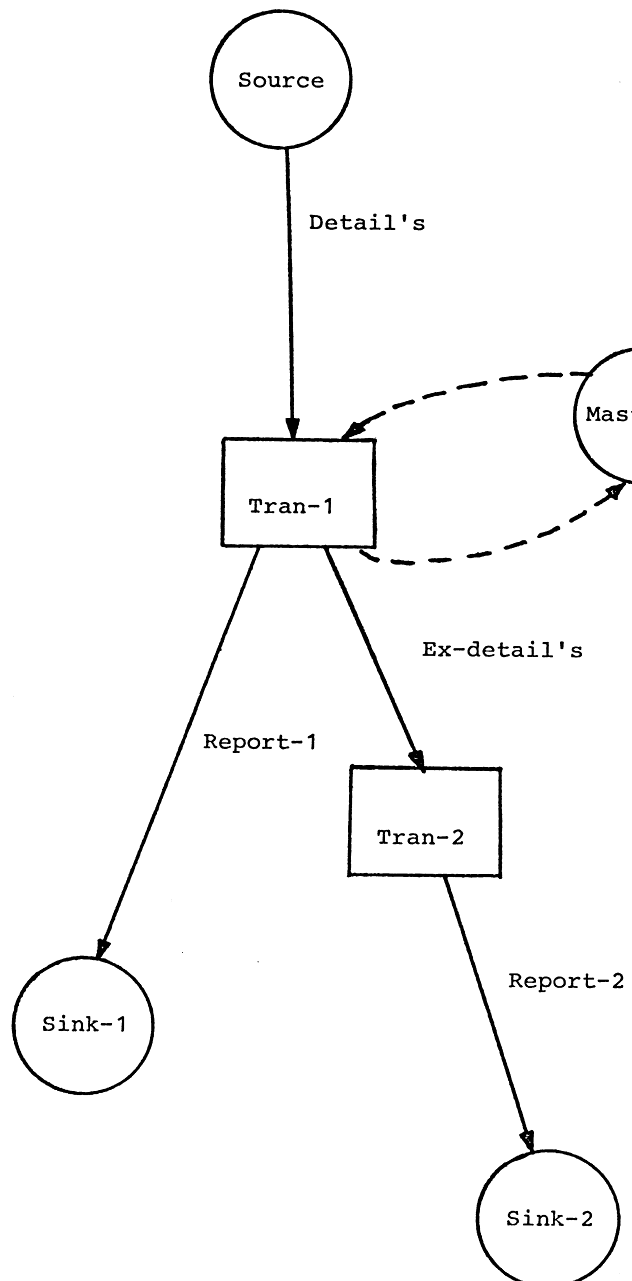

Fig. 2

Downloaded from https://academic.oup.com/comjnl/article/20/1/6/340970 by guest on 19 April 2024

1 Ext detail's	ONE PER Detail	Detail's	INPUT
2 Ext price	Unit price × Quantity	Unit price	IN Product Master
		Product Master	Master WITH Master Product number = Detail Product number
		Master Product number	IN Master
		Detail Product number	IN Detail
		Detail	CAUSE OF Ext detail
		Master's	INPUT
		Quantity	IN Detail
2 Salesman	Old Salesman	Old Salesman	IN Detail
		Detail	CAUSE OF Ext detail
2 District	Old District	Old District	IN Detail
		Detail	CAUSE OF Ext detail

Fig. 3

1 Report-1			
2 P's	ONE PER Grouped Detail's	Grouped Detail's	Detail's WITH COMMON Old Product number
		Old Product number	IN Detail
		Detail's	INPUT
3 Product number	Old Product number	Old Product number	COMMON IN Grouped Detail's
		Grouped Detail's	CAUSE OF P
3 Product total	Unit price × SUM (Quantity's)	Unit price	IN Product Master
		Product Master	Master WITH Input Product number = Product number
		Input Product number	IN Master
		Master's	INPUT
		Quantity	IN Grouped Detail
		Grouped Detail's	CAUSE OF P
3 Ytd sales	Old Ytd sales + Product total	Old Ytd sales	IN Product Master
		Product Master	Master WITH Input Product number = Product number
		Input Product number	IN Master
		Master's	INPUT

Fig. 4

1 Report-2			
2 D's	ONE PER Input Detail's	Input Detail's	Ext detail's WITH COMMON Old District
		Old District	IN Ext detail
		Ext detail's	INPUT
3 District	Old District	Old District	COMMON IN Input Detail's
		Input Detail's	CAUSE OF D
3 District total	SUM (Ext price's)	Ext price	IN Input Detail
		Input Detail's	CAUSE OF D
3 S's	ONE PER Grouped Detail's	Grouped Detail's	Input Detail's WITH COMMON Old Salesman
		Old Salesman	IN Input Detail
		Input Detail's	CAUSE OF D
4 Salesman	Old Salesman	Old Salesman	COMMON IN Grouped Detail's
		Grouped Detail's	CAUSE OF S
4 Salesman total	SUM (Ext price's)	Ext price	IN Grouped Detail
		Grouped Detail's	CAUSE OF S
2 Month total	SUM (Ext price's)	Ext price	IN Ext detail
		Ext detail's	INPUT

Fig. 5

Aggregate operations

In data processing we typically deal with aggregates such as sets or groups of entities. We will use the example application to illustrate five types of aggregate operations. In Fig. 4, a prototypical element of the group named *P*'s is defined to be a function of a prototypical group called Grouped Detail's. The mapping is a transformation from groups to documents and is indicated by the syntax ONE PER Grouped Detail's, where the group named Grouped Detail's is called the causing element.

The second type of aggregate operation is shown in the definition of Grouped Detail's. Grouped Detail's is defined to be Detail's WITH COMMON Old Product number. This operation, which was inspired by the 'Glump' in the information algebra, groups together all Detail's with the same product

Downloaded from https://academic.oup.com/comjnl/article/20/1/6/340970 by guest on 19 April 2024

number, and forms a partition of the input group Detail's. This mapping is from groups to aggregates of groups, where the member of the range group is an element of the partition. The adjective Grouped is used here to distinguish between the result of the WITH COMMON operation and one of its arguments.

The next example is an aggregate function SUM shown in the specification of Product total. This function adds up the elements of a group of scalars. The mapping which is a transformation from groups to scalars is applied to the group called Quantity's which is defined in the corresponding logical line. See the discussion on the definition of this group below. Another example of a mapping from groups to scalars is the definition of Old Product number as COMMON IN Grouped Detail's. This operation selects the Product number field in an arbitrary (since the value of this field is the same in each element of the group) member of Grouped Detail's and returns the scalar value.

The final example of an aggregate operation is given in the definition of Quantity as IN Grouped Detail. Although the IN operator normally acts like a simple selection function (see, for example, the definition of Unit price as IN Product Master), here it acts like an aggregate operation because of context. This transformation selects the value of the Quantity field in each document of Grouped Detail's and aggregates the scalar values to form a new group called Quantity's. The IN operator therefore acts like a projection function in this case.

The foregoing examples do not exhaust the range of possibilities for aggregate operations.

Associative referencing

Associative referencing involves searching an aggregate for members satisfying a given condition, but the search is the operational result of specifying the data access implicitly. Associative referencing is a commonplace feature of most data retrieval languages. The definition of Product Master as Master WITH Master Product number = Detail Product number in Fig. 3 shows how the boolean condition is stated and illustrates the use of adjectives to distinguish between the same field names in two different records. The use of singular names (Product Master and Master) indicates that a match is expected on a unique record. If more than one record satisfies the condition, an error indication will be raised. If the expectation is that multiple records will satisfy the condition, the plural syntax ('s) is used.

Elimination of arbitrary sequencing—data dependencies

There seems to be a close correspondence between procedural programming, side effects and explicit sequencing. If a program satisfies the 'single assignment' property (Tesler and Enea, 1968) (if no variable is assigned values by more than one statement), then the order of statements is irrelevant and the correct sequence can be determined by dependency analysis. Our model satisfies this property where the statements referred to above correspond to the logical lines of the pattern. In Fig. 4, the field Product total is specified before its use on the following logical line. The program is therefore in the conventional form with specification occurring before usage. However, the lines specifying Product total and Ytd sales can be interchanged without affecting the result. This outcome is guaranteed by the absence of side effects and the functional character of the step. The use of Product total without a corresponding definition indicates that its value is to be taken from the field of the same name on the document being produced.

The ability to define the lines of a pattern in any order greatly enhances the problem solving process. It should be noted that, regardless of the order of lines in a particular program, the order of lines for the purpose of output can be defined by a special pattern or template declaration associated with the document. Incidentally, the programs in Fig. 4 and Fig. 5 define the outputs to be singleton groups (they have no causing elements) although each contains a group as a substructure.

Conclusions

We have tried to show how the use of certain language features can help in the description of data processing problems. The most important of these are the use of operations on data aggregates, the elimination of arbitrary sequencing, both as to dependencies of data transformations and flow properties of the application, associative referencing, and the use of patterns to represent computations on prototypical elements of aggregates. The following statement nicely expresses the principle of lack of sequencing contraints: 'dependencies that are not present in the application itself should not appear in the program representing the application' (Goldberg, 1975).

The purpose of this paper has been to present behavioural characteristics of a language rather than exhibiting a precise description of syntax and semantics. However, the programs in Figs. 3, 4, and 5 do exhibit a particular concrete syntax. An experimental prototype of a language possessing the attributes discussed and called the Business Definition Language* (Hammer, Howe and Wladawsky, 1974) has been implemented. Since such a language reflects the entities and abstractions that are relevant to business applications it does not cater, at least at the language level, to considerations of execution efficiency. In fact, the principle of locality of reference whereby names are known only locally practically ensures the frequent appearance of common subexpressions; the treatment of such redundancies lies in the domain of an optimising compiler.

The sum and substance of the approach described here is to make data processing applications easier to specify, modify and verify by raising the level of application description.

Acknowledgements

The current tabular format and naming features of the language are due in large part to the ideas of V. J. Kruskal. W. G. Howe and I. Wladawsky read an early version of the paper and made valuable suggestions.

Downloaded from https://academic.oup.com/comjnl/article/20/1/6/330970 by guest on 19 April 2024

*The language actually consists of three components: the Form Definition component, Document Flow component, and Document Transformation component; the prototype only implements the latter.

References

CODASYL Language Structure Group (1962). An Information Algebra Phase I Report, *CACM*, Vol. 5, No. 4, pp. 190-204.

GOLDBERG, PATRICIA C. (1975). Automatic Programming, in *Lecture Notes in Computer Science*, Vol. 23, Springer-Verlag, New York.

HAMMER, M. M., HOWE, W. G. and WLADAWSKY, I. (1974). An Interactive Business Definition System, Proceedings of a Symposium on Very High Level Languages, *SIGPLAN Notices*, Vol. 9, No. 4, pp. 25-33.

HAMMER, M. M., HOWE, W. G., KRUSKAL, V. J. and WLADAWSKY, I. (1970). A Very High Level Programming Language for Data Processing Applications, to be published.

LEAVENWORTH, B. M. and SAMMET, JEAN E. (1974). An Overview of Nonprocedural Languages, Proceedings of a Symposium on Very High Level Languages, *SIGPLAN Notices*, Vol. 9, No. 4, pp. 1-12.

MCCRACKEN, D. D. and GARBASSI, U. (1971). *A Guide to Cobol Programming*, John Wiley & Sons, New York.

TESLER, L. G. and ENEA, H. J. (1968). A Language Design for Concurrent Processes, *Proceedings SJCC*, Vol. 32, pp. 403-408.