AN INTERACTIVE BUSINESS DEFINITION SYSTEM

by

M. M. Hammer*
W. G. Howe
I. Wladawsky

Computer Sciences Department
IBM Thomas J. Watson Research Center
Yorktown Heights, New York, 10598

Abstract: This paper presents a structured approach for describing business applications and a programming language that embodies the approach. This language provides business oriented building blocks for formally specifying the applications. The language is the nucleus of a Business Definition System in which users define their problems by interacting with application models in a question-answer mode.

I Introduction

Recently there has been considerable interest in developing methodologies which permit the specification of applications with minimal assistance from professional programmers (1,2,3). The reasons for this interest are varied. From a technical point of view, it is important to understand the possibilities and limitations of human interaction with computers in the human's terms. Furthermore, the dramatic shift which is taking place in the relative costs of hardware and software implies that the use of computers could be greatly expanded were it not for the prohibitive costs of developing and maintaining programs.

High level, problem oriented programming languages will certainly aid in the specification of business problems. But it remains to be seen whether non-programmers, especially in the rather unstructured commercial area, are able to cope with the rigidity of formal programming interfaces.

Interactive definition of applications, in which users answer questions and make statements in their own terms, does not require much knowledge of programming concepts. Such systems require that a model of the interaction be somehow built into them, with the facility of prompting the user for responses when necessary and associating the appropriate actions with the user responses. Knowledge acquisition systems that allow the user to hold a general and informal dialogue with the computer are being studied (1,4,5), but their use in defining real applications is not foreseen in the near future. On a more limited scope, models have been written to permit programming in general purpose languages by prompting for the program fragments (6,7). This is of limited use to the non-programmer. More recently, models have been produced that permit the interactive definition of applications for simulation purposes (8,9,10).

In the business area, the IBM Application Customizer Service (ACS) (11,12) permits the specification of some business applications, such as billing, accounts receivable, inventory control and sales analysis, by filling out a standard pre-printed questionnaire. Thus ACS is not interactive. If the ACS models do not do not fit a particular user, the resulting RPG application has to be modified, which is a rather costly and error-prone process.

The work described in this paper is an attempt to facilitate the interactive definition of business applications. The system has two main goals in mind. The first is to provide an environment in which experts can produce and modify application models in a more natural and easier manner than has so far been possible. In this context, model is simply defined to mean program fragments incorporating all commonly used options, along with an interactive user oriented questionnaire. The second goal is to permit the users, in the course of the interaction, to introduce options which have not been offered by the model and which are required in their particular business system. These points are further

discussed in the next section.

II General Approach

Producing a model for an application area is itself a formidable task. Consequently, application models should be ever-changing, reflecting the increased knowledge and understanding of the application area that will surely accompany the actual use of the model. Any successful modelling strategy has to deal with the problem of writing and modifying the models, particularly if application experts rather than traditional programmers are going to design the models.

The most important way of helping the model builder is to provide him with a language for specifying the business applications such that the options specified in the questionnaire correspond rather naturally to local units of the language. This avoids situations in which the sequences of code necessary to implement an option are scattered all over the program, and therefore, interact in a complicated way with other previously defined sequences. Clearly, a good programming language for building models must isolate and embody the right set of application building blocks.

The language must also be highly structured to permit different sections of programs resulting from selections of different options to fit together easily. This requirement demands that the language have very strict rules concerning how building blocks fit together, i.e., be highly formal, for only then can it be detected at definition time whether the different options do indeed fit together and if not, appropriate action can be taken. In addition, the number of different building blocks should be as small as possible, to facilitate the analysis of programs for consistency and their eventual interpretation or compilation.

It is important that users defining their applications understand the consequences of their answers to questions, especially if they are allowed to reject options and specify their own. This is possible if models are built in a rather disciplined and stylized way which is explained to the user, rather than simply asking a barrage of questions with no apparent connection. The underlying modelling language should encourage and facilitate such an approach to building models.

The language must satisfy two requirements to permit users to introduce their own options. First is the aforementioned locality of definitions, i.e., the user introduced options must be self-contained pieces of code, producing no major side effects in the rest of the applications. Second, the syntax of the language should be business oriented, to make it easier for users to express their individual definitions.

To summarize, in order to produce a modelling system satisfying our goals we must first define a structured, stylized approach to defining business applications which is as natural as possible to businessmen, and this approach must then be embodied in a compact, formal programming language. The rest of this paper deals with the characteristics and definition of such a language. The model building process itself will be described in future papers.

III Characteristics of Business Languages

The goal of all problem oriented specification languages is to provide concepts and rules that match the way people understand their applications. In areas where the application concepts are universal and well understood, special purpose languages have been quite succesful (13,14,15). In the business area, on the other hand, specification languages are very hard to define because there is no general agreement as to the basic conepts and methods used in describing business applications.

A number of business specification languages exist or are being defined and they each embody the designer's understanding of business requirements. Recent survey papers by Teichrow (16) and Couger (17) analyze these languages and highlight their similarities and differences.

The requirements for the application language described in the previous section are somewhat different from most of the current specification languages. The Information Algebra (18), for example, defines formal operations for describing business functions, but it does not provide a user orienter language for defining problems. The Problem Statement Language (PSL) (19,20), on the other hand, provides a complete interface in which all aspects of a business system can be defined using business oriented terms. PSL is a very rich language containing a large number of concepts. The application language defined in the next section is closer to a conventional programming language than PSL is, with a small number of building blocks from which the business terms are defined. It thus offers a more limited and structured

way of defining the business applications than most specification languages, PSL included. This is a requirement imposed by modelling.

IV The Business Definition Language

The Business Definition Language (BDL) is a specification level programming language which not only meets the aforementioned criteria but can also be compiled into executable code. BDL is composed of five (5) major components: The Document Flow Component, The Document Transformation Component, The Document Definition Component, The Human Interaction Component, and The Device Linkage Component. We will briefly indicate the basic constituents of each of these components. This will be followed by a more detailed definition of the first two.

The Document Flow Component (DFC) is used to define the basic structure of the application in terms of the organization of the business and the information flow within that organization. This structure is defined by graphically representing the documents involved in the application and the organizational units within the business that use these documents. In BDL, the organizational units are called Steps. Steps correspond to departments, sections, clerks, etc. They are connected by paths which carry the appropriate documents. For example, a typical organization might include a subset such that the Sales Order Department generates, upon receipt of an order, an invoice and a ledger copy which are forwarded to the Billing Department. The DFC representation would be:

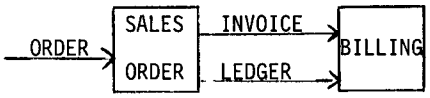

Figure 1

That is, the Sales Order step receives ORDERs and generate INVOICEs and LEDGERs which are then sent to the Billing step.

A step can be defined in terms of other steps, to reflect an organization in which departments are composed of other subdepartments. Such steps are called organizational steps.

Once the flow of information and the business organization have been specified, the details of the processing steps are defined by the Document Transformation Component (DTC). The DTC is a tabular language that is used to define those steps which produce one or more groups of documents from one or more groups of other documents. DTC steps differ from organizational steps in that they cannot be conveniently further decomposed into other application meaningful steps. Each output of a DTC step is defined by a table. Each row of the table corresponds to a single field or collection of fields that appear on the document being produced. The columns of the table subdivide the definition of each field into relevant and convenient categories, each of which has a prescribed function and a limited set of possible operators.

The Document Definition Component (DDC) is used to define the structure and attributes of each of the documents in the application. Each of the fields is listed with respect to structural associations with the other fields on the same document. For example, the fact that a Line Item on an Invoice is composed of an Item, Quantity, Price etc. is a structural association. The DDC is also used to define the format of the document in terms of the media that will eventually be used for the external realization of the document. Furthermore, the DDC will be used to define any domain restrictions that might be present, e.g. Dollars, Date, Number, etc., or any range expression that might be composed from these domains.

In any business, there will also exist a need for periodic human interaction with the executing application. For example, many source documents will require human approval at various stages. The expectations and restrictions on this human intervention will be defined by using The Human Interaction Component (HIC).

Finally, the application definition will be completed by defining a linkage between the logical media described by the source documents and the physical media which will actually be used. For example, a Point-Of-Sale application might use a magnetic coded wand reader in conjunction with a keyboard in order to input a Line Item consisting of Item # and Quantity. In this case, two different physical media are used to link with a single logical medium. This linkage will be defined by the Device Linkage Component (DLC). With this brief description of each of the components of BDL in hand, we will provide more detail for the Document Flow and Document Transformation components, since these are the more interesting and novel sections of BDL.

A The Document Flow Component

As previously stated, the DFC is used to define the basic structure of the application in terms of the flow of information and organizational units of the business. The DFC is a graphical language which is composed of Steps, Paths, Documents, and Files. In BDL, Files are simply a collection of instances of Documents. Therefore, there is a different File for each potentially persistent type of Document.

The application designer then draws an information flow graph by connecting the Steps by Paths and indicating the Files that are to be accessed by these Steps. This graph can be easily generated in a top-down fashion following the organization of the business. For example, the previous example showed connections between the Billing Step and the Sales Order Step. If the Sales Order Step can be further decomposed into other Steps such as Order Entry and Produce Invoice, then the DFC graph can be enhanced to accommodate this further definition, as shown in Figure 2. This top-down definition technique continues until no further decomposition is possible.

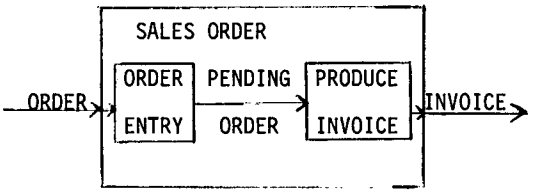

Figure 2

Eventually, the application designer reaches a point at which no further decomposition is possible. These Steps are then defined by using the DTC language and are called DTC Steps. A DTC Step is available for execution when all of its inputs are present. Whenever it executes, a DTC Step produces all of its outputs "simultaneously". Each Path contains an intrinsic queue so that the inputs to DTC Steps can be coordinated.

Files are represented in BDL by a circle and a File access by a dotted line as in Figure 3 where CM represents the File.

Whenever ORDERs arrive at the Order Entry Step and it executes, the entire contents of the CM File are made available and the Step produces Pending Orders. The synchronization between several different Steps accessing the same File is solved by permitting only one step to update a File at any given time. Since the entire File is locked for update any other access for update must be delayed until the first update is completed. However, there is no reason to prevent read-only accesses. This is precisely the semantics of File access in BDL.

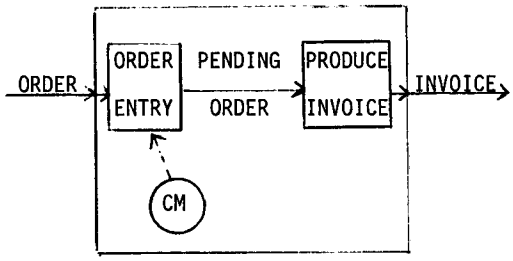

Figure 3

As will be seen later, the DTC language will be oriented around aggregate operations. This orientation also suggests that the input to these Steps should be groups of documents whenever possible. The semantics of the DFC language are that Paths carry groups of Documents. In a transaction oriented system the groups on certain paths may degenerate to singleton groups but this will not cause any burden on the application designer as he will not have to be overly concerned with the distinction between singleton groups and individual Documents.

The DFC language also contains several special purpose Steps to aid in the definition of the information flow. For example, an Accumulate Step is included to collect small groups of Documents and form larger groups out of them, e.g. collect daily groups of ORDERs for a week to produce a weekly summary report. Similarly, a Stream Step performs the inverse operation. A Copy Step duplicates the group and a Join Step merges groups by time of arrival.

Several other special purpose Steps also exist but they will not be enumerated here. Once the document flow has been completed to a point such that no further decomposition exists, the application designer is ready to fill in the details of the application by writing all of the applicable DTC tables.

B The Document Transformation Component

The Document Transformation Component (DTC) is used by the application designer to define the generation and use of the Documents that constitute the application. Each DTC Step in the application will produce one or more groups of Documents as output. The definition of a DTC Step is oriented

around the concept of defining the output in terms of the input. This is accomplished by completing a tabular definition for each of the different outputs of the DTC Step.

The definition of a DTC table begins by listing the fields that will appear on the associated Document along with the structural association of these fields. Some of the fields are single valued while others are repeating structures. Repeating structure fields are indicated by the keyword Group following the field name. The structural associations are indicated by indentations. A typical list of Document field names appears in Figure 4.

```
INVOICE
  CUSTOMER NUMBER
  CUSTOMER NAME
  CUSTOMER ADDRESS
  ITEM GROUP
    ITEM NUMBER
    ITEM COST
    QUANTITY
    AMOUNT
    TAXCODE
  DISCOUNT
  TOTAL
  TOTAL TAXABLE
  TOTAL DUE
```

Figure 4

Before defining the rest of the DTC, we must digress for a moment. As stated previously, the paths carry groups of Documents. These groups are then the input to DTC steps and other groups of Documents are the output. The application designer defines a DTC step by specifying the transformation between the input groups and the output group. This is accomplished by focusing on a prototypical element of the output group and specifying completely how this single Document is produced from a subset of the input group. This specification can then be used as a template and all of the required output documents can then be produced from an input group. This method is analogous to the definition of a functional mapping from one set to another, in that a general function is given for a prototypical element of the output set in terms of the input set.

In DTC the functional mapping corresponds to the definition of a table. As with functional mappings, one must define the conditions under which any given element of the output is produced. For example, one element of the output group might be generated for each element of the input group. Another possibility would be to have one element of the output group generated for each element of a partition of the input. In this case, the definition would also have to contain a specification of the partitioning.

In the DTC we call this concept Causality. For each DTC table we have to specify the causality of the Documents that are to be produced by the DTC step. This is done by first indicating the name of the input group to be used, followed by an indication of the conditions to be used in determining which subgroup of that input group is to be used for the generation of the prototypical output element. This subgroup is called the Causality Sub Group.

The Causality of a group of Documents is specified by filling in the Domain, Group and Filter columns of a table next to the name of the document, as shown in Figure 5.

An [Output Group Name] is the name of the group of Documents that are to be generated by the execution of this DTC step. A [Group Name] is the name of any group available within this Step. In general, this will be the name of one of the input groups. A [Group Expr.] is the specification of a subsetting function to be applied to the Domain. The most common [Group Expr.] is to name some field within the Domain Group. This causes the Domain Group to be partitioned by the unique values of that field name. For example, the production of one Invoice for each customer who had made an Order would be specified as in Figure 6.

One other possibility for specifying the Causality of the group of Documents is to explicitly list a set of range expressions for a field in the Domain. In this case, the subsetting function does not, in general, constitute a partition as in the previous specification since overlapping range expressions may generate overlapping subroups. Other possibilities exist but they will not be enumerated here.

The causality of a group of Documents defines the subgroup of the input group that is to be used for the production of each of the Documents and the cardinality of the output group. Once the causality has been defined, the application designer now focuses on a prototypical Document and uses the Causality Sub Group for that Document to define the rest of the fields on that document.

In order to define a single valued field, the application designer must define what information is needed to compute a value and the computation that

NAME	DOMAIN	GROUP	FILTER
[Output Group Name]	[Group Name]	[Group Expr.]	[Boolean Condition]

Figure 5

NAME	DOMAIN	GROUP	FILTER
INVOICE GROUP	ORDER GROUP	CUSTOMER #	

Figure 6

NAME	DOMAIN	GROUP	FILTER	DERIVATION

Figure 7

is to be performed. The information needed to compute a value for a given field is called the Dependency of that field. In most cases, the Dependency of a field is simply the Causing Sub Group. However, in general, the Dependency of a field can be any group that is constructable within this DTC Step. Therefore, the same three columns that were used for the specification of the Causality are also used for the specification of the Dependency. In the case that the Dependency is simply the Causing Sub Group, the abbreviation CSG is written in the Domain column next to the name of the field being defined.

Once the Dependency for a field has been defined, the only thing that is left is the specification of the computation required to achieve a value. This specification is written in a column labeled Derivation which is juxtaposed with the Filter column. The Derivation is specified in terms of the basic arithmetic operations or aggregate operations such as SUM, COUNT, etc. Several special purpose operations have also been defined. For example, SEQ is an operation which generates the next number in a sequence. Second, the operation COM takes a field name of a group as the argument and checks the value of that field in each element of the group. If all of these values are identical, that value is selected, but if they are not all identical, an error is generated. Although this is not a complete list of the potential operations that can be used in the specification of a Derivation, it is a fair indication of the kind of operations that are planned.

The specification of a repeating structure field is identical to the specification of a Document. First, the Causality of the group is established and then the individual fields within the repeating structure are individually defined. Just as in the case of Document definition, the Causing Sub Group is denoted by CSG. However, in this case the Causing Sub Group refers to the Causality of the repeating structure and not the Causality of the Document. The Causality of the Document can be referenced by the composed name consisting of the Domain column entry in the Causality of the Document subscripted by the Document name. For example, the Causality of the Invoice as defined in Figure 6 is referenced by ORDER GROUP(INVOICE). This naming scheme can be used to reference any Causality that has been defined within the scope of the structural associations of a field.

The only remaining feature of the DTC is the Conditional Derivation. In some cases, it may be necessary to select from several different Derivations for the same field. For example, several different rates may be used to compute the discount and the appropriate one is chosen as a function of the customer classification that is associated with each ORDER. This is indicated by listing the boolean expressions next to the appropriate Derivation in the Derivation column for that field. The boolean expressions are separated from the derivation by a solid line. A conditional derivation is evaluated as a McCarthy conditional.

The standard DTC table is shown in Figure 7

V Example

The basic ideas of the DTC can be

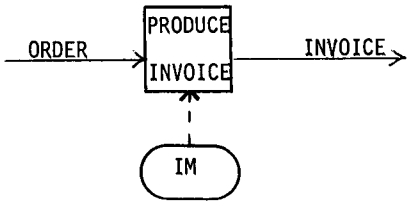

Figure 8

ORDER GROUP
  CUSTOMER #
  CUSTOMER NAME
  CLASS
  CUSTOMER ADDRESS
  ITEM GROUP
    ITEM #
    QUANTITY

IM GROUP
  ITEM #
  PRICE
  TAXCODE

Figure 9

	NAME	DOMAIN	GROUP	FILTER	DERIVATION
1.	INVOICE GROUP	ORDER GROUP	CUSTOMER #		
2.	CUSTOMER #	CSG			CUSTOMER #
3.	CUSTOMER NAME	CSG			CUSTOMER NAME
4.	CUSTOMER ADDR	CSG			CUSTOMER ADDR
5.	ITEM GROUP	CSG	ITEM #		
6.	ITEM #	CSG			ITEM #
7.	ITEM COST	IM GROUP		IM.ITEM # = INVOICE.ITEM #	IM.PRICE
8.	QUANTITY	CSG			SUM(QUANTITY)
9.	AMOUNT	INVOICE			ITEM COST x QUANTITY
10.	TAXCODE	IM GROUP		IM.ITEM # = INVOICE.ITEM #	IM.TAXCODE
11.	DISCOUNT	CSG			CLASS=A | .1 x TOTAL CLASS=B | .05 x TOTAL CLASS=C | 0
12.	TOTAL	INVOICE			SUM(AMOUNT)
13.	TOTAL TAXABLE	INVOICE		TAXCODE = "*"	SUM(AMOUNT)
14.	TOTAL DUE	INVOICE			TOTAL - DISCOUNT

Figure 10

shown in terms of an example that produces INVOICEs from ORDERs and accesses the IM (Item Master) File, as shown in Figure 8. Figure 9 gives the definition of the ORDER and IM documents. This is a simplified example that does not accommodate the inspection of the Item Master to determine if the quantity-on-hand is sufficient. This can easily be done in a previous step and the unfillable ORDERs can be routed to a different step. Therefore, we will assume that all ORDERs can be filled. The Document Transformation table for INVOICEs is shown in Figure 10.

Line 1 The Causality of the INVOICE is specified as one INVOICE for each customer making an ORDER.

Line 2,3,4 The appropriate values are retrieved from the Causing Sub Group of ORDERs.

Line 5 The Causality of the ITEM is specified as one Item for each unique Item on the ORDER. since a customer may make separate ORDERs for the same item, this prevents duplicating that item on the INVOICEs.

Line 6 The appropriate value is retrieved from the Causing Sub Group of Items.

Line 7 The Price of the Item is retrieved from the IM file by locating the appropriate record with the Filter.

Line 8 Since there may have been more than one ORDER for the same item by the same customer the Quantity field is summed to get the correct value for the INVOICE.

Line 9 The amount is computed from fields in the INVOICE.

Line 10 The Taxcode is also retrieved from the IM file.

Line 11 The Discount is computed as a Conditional Derivation from the Class that has been assigned to the Order. Note that the discount can be specified as a function of the

Total even though the Total field is specified later. These data dependencies can be sorted out by an intelligent translator.

Line 12 The Total is computed from the Line Items being INVOICEd.

Line 13 The Total Taxable field is computed from those line items that are marked to be taxable. This marking consists of an '*' in the Taxcode field.

Line 14 The Total Due is computed from the appropriate fields on the INVOICE.

After the completion of the definition of the Documents involved in terms of some appropriate source and sink devices, an executing program can be generated that will accept fillable ORDERs and produce INVOICEs.

VI Conclusions and Extensions

In this paper, we have presented an overview of a formal specification level programming language intended for application designers who are not professional programmers but are familiar with the interactions of the application and are capable of formally representing these interactions. This language will now be used as the basis for the design of an advanced modelling and customization system, satisfying the needs of application development for smaller or specialized businesses.

Acknowledgements

The authors wish to thank V. Kruskal, B. Leavenworth, C. Lewis and D. Lomet for their helpful suggestions in preparing this paper and their continued assistance in the research effort.

References

*. Present address: Project Mac, M. I. T., Cambridge, Mass.

1. Balzer, Robert, Automatic Programming, Technical Memo, Information Sciences Institute, University of Southern California, September, 1972.

2. Martin, William, et. al., Automatic Programming Internal Memos, 1972, 1973.

3. Automatic Programming Workshop, M.I.T., January, 1973.

4. Feigenbaum, E., Buchanan, B. and Lederberg, J., "Generality and Problem Solving! A Case Study Using the DENDRAL Program" in Machine Intelligence 6 (B. Meltzer and D. Michie, eds), American Elsevier, 1971.

5. Winograd, T. Understanding Natural Language, Academic Press, New York and London, 1972.

6. Ginsberg, A. S., Markowitz, H. M., and Oldfather, P. M., "Programming by Questionnaire", The Rand Corporation, RM-4460-PR, 1965.

7. Low, D. W., "Program/Text Generation: A Decision Table Approach", IBM Data Processing Division, 320-2633, December, 1969.

8. Heidorn, G. E., "Natural Language Inputs to a Simulation Programming System", Doctoral Dissertation, Naval Postgraduate School, Oct. 1972.

9. Oldfather, P. M., et. al. "Programming by Questionnaire: The Job Shop Simulation Program Generator", The Rand Corporation, RM-5162-PR, 1967.

10. Connors, M. M., et. al., "The Distribution System Simulator: Overview", IBM Data Processing Division, 320-2630, December, 1969.

11. IBM Application Customizer Service, Sales and Distribution Accounting Application Manual - Form No. SH20-1004.

12. Application Customizer Service Sales and Distribution Questionnaire - Form No. S320-1040-2.

13. Falkoff, A. D., Iverson, K. E., "APL/360 Terminal System", Proceedings of the Symposium on Interactive Systems for Experimental Applied Mathematics, Academic Press, New York (1968).

14. Brown, S. A., Drayton, C. E., Mittman, B., "A Description of the APT Language", Comm. ACM, Vol. 6, No. 11, (Nov. 1963).

15. Efron, R., et al, "A General Purpose Digital Simulator and Examples of its Applications, pts. I, II, III, and IV", IBM Systems Journal, Vol. 3, No. 1, (1964).

16. Teichroew, D., "A Survey of

Languages for Stating Requirements for Computer-Based Information Systems", Fall Joint Computer Conference, 1972.

17. Couger, J. D., "Evolution of Business System Analysis TECHNIQUES", Computing Surveys, Vol. 5, No. 3, September, 1973.

18. CODASYL Language Structure Group, "An Information Algebra Phase I Report", CACM Vol. 5, No. 4(April, 1972).

19. Hershey, E. A., et. al., PSL/II Language Specifications, Version 1.0 ISDOS Working Paper No. 68, University of Michigan, Dept. of Industrial and Operations Engineering, Ann Arbor, Michigan (Feb. 1973).

20. Teichroew, D., Sayani, H., "Automation of System Building", Datamation, August, 1971.