Natural language inquiry to an open-ended data library

by GEORGE W. POTTS

Meta-Language Products, Inc.
New York, New York

INTRODUCTION

New technologies often create an effect similar to children bragging about their new Christmas toys. This “me-too-ism,” as it exists in the proprietary software community, is unfortunate because it robs credibility from this emerging industry at a very crucial time. In this paper a new computer system and language, “MUSE,”* is presented with the intention not to follow this pattern.

“MUSE” could be said to belong to that family of languages called “non-procedural” in that it is not necessary to produce a sequential flow of programming logic to force output. This is a somewhat ambiguous concept in that “MUSE” does incorporate a capability for the user to embed “procedures.” It is expected that this (among other features) will enable “MUSE” to be used as an information system for management as well as other disciplines that require easy access to and manipulation of large volumes of data.

In the development of “MUSE” an attempt has been made to refrain from producing anything of merely academic interest. It is the child of a very informal set of circumstances in the authors’ experience which has consistently shown the need for an unstructured dialog between those who have non-routine problems and the computer that can contribute so much toward a solution.

BACKGROUND

Among the spectrum of new technologies four seem to be travelling convergent courses. They are:

—Time-sharing
—Data management
—Management information systems
—Natural programming languages

*“MUSE,” an acronym for Machine-User Symbiotic Environment, is a trademark of Meta-Language Products, Inc.

Each of these subjects has had a good bit of exposure in computer and business publications recently... much of it groping. There follow brief discourses, not explanations, on the above subjects with an attempt to offer a few preparatory insights. Following that, there is a description of “MUSE” and a few of its more interesting features that tie these subjects together.

Time-sharing

Computer time-sharing, although it provides remote access and real-time capability, is primarily a medium for nonroutine data processing. Here the interactive environment is the message. It is not where or when creative interaction takes place, it is that it take place.

A good bit of controversy exists just because of confusion on this first point. It is not surprising that the cult of time-sharing purists take issue when a remote polling capability is called “time-sharing.” Polling algorithms are created to provide routine, low-level input and inquiry. The advantage of true time-sharing is creative interaction. The system designers who have diligently labored to develop a general purpose time-sharing capability must be complimented on their restraint when their work is confused with a reservation system or message switching.

Another point of confusion regarding time-sharing is due to a changing optimization emphasis. The entire orientation of batch processing is to push jobs through the computer as fast as possible. In time-sharing, as Dartmouth’s Dr. Kemeny likes to point out, the optimization is more for the user. It is very difficult for some to accept the premise that machine time should be wasted so that optimum use can be made of the ma-
chine-user pair. The motivation of some of the recent studies of comparative productivity between pro-
gramming in a time-shared vs. a "batch" environment appears tenacious in the very same sense. While cur-
tently the worst case might be only slight increases in productivity, the potential of this type of creativity is huge using time-sharing while with batch it has about run out.

There are at least three major types of effort that lend themselves to the creative use of time-sharing:

1. Preprogrammed aids for scientists, engineers, etc.
2. The development and debugging of computer pro-
grams for both the batch and time-shared environ-
ments.
3. Information systems for inquiry and interpretation.

Notice that only the first of these has been exploited by the time-sharing vendors. The second use is probably a major objective of the large computer networks cur-
rently under construction. The last use is the reason for this paper.

Natural programming languages

English or "natural" programming languages have had a somewhat orderly development from the first halting steps, using mnemonics for binary machine codes, through FORTRAN and COBOL to the "na-
tural" languages. It is interesting to note that BASIC, a language only slightly simpler than FORTRAN, is used by over 60% of all users of time-sharing.

Languages that had to be debugged in a batch processing environment developed quirks of form that have a remarkable staying power in the languages now used in time-sharing. Slowly the older languages are being modified to permit interactive debugging without eliminating the compiler phase. This permits the compiled code still to be run in batch. Interpretive languages have gained popularity commensurate with the increased use of time-sharing. These are languages that are not reduced to a machine language level but rather to encoded forms that are never given machine control but are interpreted by other operations code. Interpreted code is usually slower running than compiled code and thus never had much use in batch shops. Not surprisingly, however, the first time-sharing lan-
guages interpreted are very similar to those that were formally compiled (e.g. QUIKTRAN). The advantage of both interpretive languages and interactive compila-
tion is that they make error detection and cor-
rection a creative and involving experience.

The current step in language development appears to be the attempted elimination of all code sequences except the problem statement itself (e.g. APL). With refinement the user ignores data input/output detail, compiler directives and any explicit statement of the sequential flow of logical events within the machine. Here then is the threshold of real natural language interaction. As the problem statement can be made to look more like English, so can the interactive experience be made more universal.

Data management

Data has been referred to as the fifth economic factor of production (as vital as land, labor, capital and management).* It is clear that a great deal of this data is numeric information and that manipulation of this data with computers is becoming indispensable in institutional operations.

The management of this data, or their organization for quick and easy reference, is a discipline with a checkered past. It would not be stretching a truth too far to say that data processing has followed the course determined by the development of hardware for interfacing with external data files. Fortunately for the computer industry the first technology of data referring, sequential access, is most easily applied to routine data processing (that with the most obvious cost benefits). The other major mode of data referencing, direct access, has matured along with the growth of time-sharing. Now direct access devices are becoming available with data rates, capacities and reliability far more suited to nonroutine, unplanned data referencing.

Again, notice the subtle misapplication of this new potential due to lagging system software development. Many languages in time-sharing still reference data files sequentially, even though they are on a direct access device. This is forgivable if referencing small private data files, but intolerable if reference is at-
tempered to very large, common data libraries (which, conveniently, also cannot be referenced with most languages in time-sharing). It would seem that a large, open-ended, data library that can be directly accessed, simultaneously, by a large number of time-sharing users would be the answer to most of the logistics problems of data management.

The only rub is that this structure, which is so well suited for non-routine data manipulation, is unsuitable for routine data processing, where throughput rates are far more critical. A simple solution, then, would be to have two data structures, one for each orientation. In most cases, routine data processing would use sequential access (except where transaction levels are small) and

---

* I have lost this reference.
non-routine data processing would maintain its own
directly-accessed data library. The non-routine environ­
ment would accept data from the routine environment
but not, in general, return any new data in the opposite
direction. It is as though there were a semi-permable
membrane between the two structures. This suggested
solution would probably hold until far cheaper, larger
and faster direct access devices are available.

Management information systems

Management information systems might be said to
have a checkered present, as there seems to be no
obsolescence of approach. The first attempts, on com­
puters, to provide management with the necessary data
for decision making culminated in “exception report­
ing.” This resulted not because it was the best way of
capturing interesting data from the routine processing
environment, but because it was the easiest. As a
consequence, exception reporting has two major draw­
backs:

1. Someone must decide, a priori, how decisions are to
be made and so design exception reports that contain
statistics highlighting this methodology. This is a
dangerous approach because it eliminates flexibility
at the very point it is most needed.

2. To generate enough statistics to cover all eventu­
alities is to generate too much paper to be con­
veniently read. It is a common experience to see
stacks of exception reports used more as evidence
of concern than as a decision-making tool.

It should be permitted the individual manager, in
his own way, to have access to all that data that could
affect his operations and to be able to form construc­
tions from this data in a way best suited to the problem at
hand.
The recent proliferation in the market place of data
management systems must indicate an intuitive dis­
satisfaction with the old methodologies. However it is
not plain that many of them offer a new alternative.

THE SYSTEM

A computer language has now been developed that:
—exists in a time-shared environment
—uses the English sentence form as its basis for
man-machine communiation
—incorporates a simple data-capturing tool and
provides reference to a large common data
library
SIMPLE QUALIFICATION

OPTION 1) SALES FOR 1965 FOR G.M.
OPTION 2) G.M.'S SALES FOR 1965
OPTION 3) SALES FOR G.M. FOR 1965

CLASS NAMES (UNIVERSAL SETS)

SALES FOR ALL COMPANIES FOR 1965

AMERICAN CAN XXX.X
AVCO XXX.X
BETHELHEM STEEL XXX.X
CONTAINER CORP. XX.X
G.M. XXX.X
- -

SALES ($ - MILLIONS)

1965

SALES

G.M. XXX.X

Figure 2

ARITHMETIC FORMULAE

LISTS (SET CREATING)

SALES, EARNINGS FOR G.M., FORD FOR 1965

SALES + OTHER INCOME FOR G.M., FORD FOR 1965

1965

SALES

($ - MILLIONS)

G.M. XXX.X
FORD XXX.X

1965

SALES + OTHER INCOME

($ - MILLIONS)

G.M. XXX.X
FORD XXX.X

Figure 3

Figure 5
the dialog as the user wishes. However, declaratives are the only ones preserved when the dialog is recorded. Dialogs may also reference other dialogs and are identified in the same manner as any other entity in the system—up to 5 words of 10 alpha characters each.

In order to explain the data referencing and manipulative capabilities of “MUSE” there follows a series of eleven illustrations which represent the relationship between dialog and output. The text excerpts that produce the output may comprise part of a declarative or interrogative sentence. To visualize this, preface these excerpts with “INCLUDE” or “WHAT ARE” respectively. Also the output produced is very stylized and need not be of three dimensions.

In Figure 2, the word “for”, or its synonyms is used in “MUSE” as an operator to order the qualification process. This process locates data in a data library of $N$ dimensions by figuratively intersecting planes passing through identifier-located points on every necessary coordinate axis. The intersection of these planes produces a unique disk address for direct data reference.

In Figure 3, identifiers of like class may be explicitly grouped into lists. This normally produces a vector of data elements on output.

In Figure 4, these lists can be given identifiers and used implicitly (see Figure 8) or the identifier for a class of identifiers implies the universal set.

In Figure 5, normal arithmetic capability is available within a class of identifiers or between classes (using parentheses).

In Figure 6, functions are available to transform individual data elements on a one-for-one basis.

In Figure 7, vector functions are shown for sales, earnings for G.M. for the average of 1965, 1964, 1963.

In Figure 8, natural language equivalences are given, such as equating “GENERAL MOTORS INC.” with “G.M.”, “PLUS” with “+”, “N.F.C.” with “NET FOR COMMON”, and more.

From the collection of the Computer History Museum (www.computerhistory.org)
SET EXPRESSIONS

SALES FOR THE AUTO INDUSTRY COMPANIES WHICH ARE IN THE FORTUNE TOP 10 FOR 1965

<table>
<thead>
<tr>
<th>Company</th>
<th>1965 Sales ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORD</td>
<td>XXXX.X</td>
</tr>
<tr>
<td>G.M.</td>
<td>XXXX.X</td>
</tr>
<tr>
<td>CHRYSLER</td>
<td>XXXX.X</td>
</tr>
</tbody>
</table>

Figure 9

In Figure 7, functions are available to reduce a vector of values to a parameter. In Figure 8, new identifiers or identifier groupings can be equated with old identifiers or identifier/operator groupings.

SIMPLE QUALIFICATION CLAUSE (WITH RELATIONAL OPERATOR)

SALES, EARNINGS FOR 1966 FOR ALL COMPANIES WHOSE SALES FOR 1965 \( \geq \) $100 MILLION

<table>
<thead>
<tr>
<th>Company</th>
<th>1966 Earnings ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMERICAN CAN</td>
<td>XX.XX</td>
</tr>
<tr>
<td>CONTAINER CORP.</td>
<td>XX.XX</td>
</tr>
<tr>
<td>G.M.</td>
<td>XX.XX</td>
</tr>
<tr>
<td>I.B.M.</td>
<td>XX.XX</td>
</tr>
</tbody>
</table>

Figure 10

QUALIFICATION CLAUSE (INCLUDING BOOLEAN OPERATOR)

VALUE ADDED FOR 1966, 1967 FOR ALL COMPANIES WHOSE SALES FOR 1965 \( < \) SALES FOR 1966 AND EARNINGS FOR 1967 \( > \) $10.00

<table>
<thead>
<tr>
<th>Company</th>
<th>1967 Value Added ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMERADA</td>
<td>XX.XX</td>
</tr>
<tr>
<td>I.B.M.</td>
<td>XX.XX</td>
</tr>
</tbody>
</table>

Figure 11

In Figure 9, explicit or implicit lists can be combined using standard Venn criteria. In Figure 10, explicit or implicit lists can be culled for members which, with further qualification, meet a relational criteria. In Figure 11, relational tests may be combined with boolean operators.

ASSIGNMENT

PROFIT FOR 1970 FOR THE PARTS DIVISION IF THE TURNOVER IS .5%

<table>
<thead>
<tr>
<th>Division</th>
<th>1970 Profit ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARTS DIVISION</td>
<td>XX.XX</td>
</tr>
</tbody>
</table>

Figure 12

From the collection of the Computer History Museum (www.computerhistory.org)
And in Figure 12, identifiers that exist without definitions (variables) in dialogs or language equivalences may be assigned values (symbolic or numeric) with a form of the verb "to be."

Syntax analysis in "MUSE" primarily checks operator nesting structures, juxtaposition of language tokens and parenthesis balancing.

With "MUSE," then, a satisfactory data referencing algorithm seems to be available within a sentence form that seems "natural" to its user. The latitude of possible forms is expanded by inserting language equivalences into the dictionary.

Teaching module

"MUSE" permits a question and answer interaction that bears no relation to data or calculations. This is an unstructured dialog where information is requested about how "MUSE" works. The system responds with answers that also suggest other questions. In such a manner, the user picks up the "MUSE" technique at his own speed.

This feature's first implementation is very similar in approach to that taken in the HELP system developed at Berkeley. It analyzes how, when and why questions with a key word scheme and produces prewritten answers to anticipated questions or question sequences.

A sample of this form of dialog is provided (Figure 13).

Data-loading and maintenance module

This module could easily (and probably will) be the subject of an entire paper. The idea that there can be a generalized interface to the real world of data input is the dipole of the earlier concept of generalized data output with RPGs.

Data loading in the "MUSE" system is accomplished by first engaging in a unique question and answer dialog. Throughout this interchange the system is asking the questions and the user is providing the answers.

2. IS A PHYSICAL INPUT REQUIRED? YES
2.1 ON WHAT DEVICE? TAPE
2.1.1 HOW IS THE INPUT ENCODED? BCD
2.1.1.1 WHAT IS THE PARITY? EVEN
2.1.1.1 WHAT IS THE PARITY? EVEN
2.1.2 ARE THERE INITIAL (HEADER) RECORDS TO BE SKIPPED? YES
2.1.2.1 HOW MANY PHYSICAL RECORDS ARE TO BE SKIPPED? 1
2.1.2.2 MAXIMUM SIZE OF SKIPPED RECORDS? 150
2.1.3 PHYSICAL RECORD SIZE IS? 150
2.1.4 LOGICAL RECORD SIZE IS? 300
2.2 IS PHYSICAL INPUT REQUIRED FOR DATA INPUT? NO
2.3 IS DATA STRUCTURE (ALL/PART) DERIVED FROM INPUT? YES

Figure 14

An excerpt of this form of dialog is provided (Figure 14).

These systems queries must obviously be answered by persons familiar with data processing. They will be passing on to the "MUSE" system information regarding the physical form of the data file, its logical form, the identifiers used to reference the data and other attributes of the data. This is, in effect, the bulk of the documentation normally associated with every batch processing data file.

This dialog is used to actually start the loading process and convert the data into the form used by the rest of the "MUSE" system. These dialogs are preserved for the purpose of updating with similar data files.

The only assumption made by this data-loading mechanism is that the data is in computer acceptable form and that it has been formatted for data processing (as opposed to typography, for instance).

Meta-language processor

This module performs the following major functions:

- Parsing of character strings into language tokens
- Syntax analysis of operator sequences and sentence forms
- Semantic analysis using token groupings
- Encoding to the threshold of non-reversibility into text
- Idiomatic translation of synonyms and other more complex language equivalences
- Polish interpretation of fully reduced dialog structures
- External referencing of data and the resolution of other exogenous requests
- Performance of operator requirements

"MUSE" is a proprietary product which precludes a detailed analysis of system programming techniques used to accomplish the above. Suffice it to say that it is complex in an orderly way. Its authors have taken many pains to avoid a kludged construction.
Report generator

The report reproduced (Figure 15) is part of the one developed from the declarative sentences displayed under Normal interaction module. Notice the symmetric structure reflecting the dimensional ordering of output. The “for” or qualifying operator has served not only to arrange the output but also to unravel the sequence of references to the data library.

The “MUSE” user, you may have noticed, has made no statement as to what is to go where in the report, how headers are to be displayed, what units are displayed, or even where the decimal point is to be placed. These were all developed, by “MUSE,” from the dialog. There is no user intervention required to organize and produce a report from a declarative dialog. He can, however, if it be necessary, change the units and scaling of data, rotate the report axis, remove or insert extra spacing and output the report to a variety of devices.

The reason why there is such a degree of initiative on the part of the system in formatting output is that “MUSE” was designed with the assumption that it is to be used for non-routine data processing. Doing this has caused a reversal in the normal temperament of programming. Now the language is far more artistic and the output far more functional.

BUILDING BLOCK CONCEPTS

The creation of “MUSE” unequivocally depends upon the refinement of, and the belief in a small set of fundamentals. They are given here to present another lens with which to inspect the system:

Primary data

The “MUSE” system is designed to operate most effectively entirely with stored primary data. This is defined as source data, or sampling data, or data to which the construction key has been lost. It is the opposite of constructed or secondary data which are arithmetic combinations of two or more primary data elements. This does not mean that secondary data is not available through “MUSE.” It just means that it is not stored. What are also stored are the declarative procedures that can reference primary data, combine it arithmetically, and display it as though it had been stored.

The advantages of this approach are as follows:

—The primary data together with declarative procedures take far less direct-access storage than the combination of primary and secondary data.
—There is no complex back indexing necessary to adjust primary data given changes in secondary data.
—There is no chance of inconsistency of result if the primary data is updated without the secondary.
—The user has far more flexibility in changing the construction of secondary data elements, or creating new ones. In fact, he may even remove entire declarative constructions without performing violence on the system.

The dimensionality of data

The realization that most numeric data, as it is organized for data processing, can assume, in this organization, a logical structure similar to a regular N-dimensional array has been fundamental in the design of “MUSE.”

What has tended to obscure this point has been the great attention given to physical data structures. These physical structures become laborious due to the storing of textual information along with numeric, the great disparity between the size and number of the vector coordinates of the logical structure, and the size and dimension constraints of the physical storage medium.

The advantages of this logical form of data structure are:

—There is much less indexing information necessary to permit random retrieval. So much less that this indexing information can be stored on faster storage (e.g., drum vs. disk).
—This, of course, allows much faster accessing of data.
—It greatly simplifies the language needed for data referencing. For example, the qualification sequence “SALES FOR XEROX FOR 1965” is all that is needed to delimit a unique data element.
—It permits the direct loading and interfacing with the current data library of virtually any new data file.
Bootstrapping language

"MUSE" is an extendable language. When the "MUSE" system is installed it has a dictionary, with definitions, of approximately 250 entries. This dictionary is expanded as a result of two classes of activity:

1. The loading of data and the resultant adding of new identifiers, definitions, secondary data constructions, and identifier groupings.
2. The inserting into the dictionary of synonyms and more complex language equivalences.

The concept of a bootstrapping language is interesting because:

- The dictionary is almost entirely user-built.
- It provides the user the opportunity to communicate in his own idiom.
- The multitude of possible language forms can make the system seem very "forgiving."
- It permits both verbose and shorthand notation.

Information about information

In any single arithmetic computation there is really more than one thing going on. Not only are numbers being combined but also the attributes of numbers are (or should be) similarly resolved. The development of "MUSE" has taken into account this parallelism of calculation and provides the following levels of computation with every simple arithmetic operation:

- The scaling of both numbers is combined to provide the scaling of the answer.
- A count of significant digits for output is maintained.
- The units of both values are compared or combined to provide the answer units.
- Attempts are made to preserve the integer form of any numbers.

Human engineering

In the design of "MUSE" effort has been expended to make it sympathetic with its user. If non-technicians are to be brought into dialog with a computer they must be appreciated for what they are, not what they might be. The adjustment must be made in the electronics and not the emotion. The following are some of "MUSE's" features that were motivated by the above realizations:

- The translation process takes place in a series of levels. Each one notes different user errors and permits correction of these errors. This is not precisely the same as incremental compilation which translates, completely, one statement at a time. In "MUSE," each statement goes through functional phases of translation which, in some cases, may be separated in time.
- "MUSE" provides full line, word, and character editing capabilities at all levels of the dialog process.
- The syntax of commands permits construction of user requests which expand on the standard VERB-OBJECT form.
- The "MUSE" system from time to time will give suggestions or warnings to the user. These are not errors, but potential errors.
- The "MUSE" system uses the Teletype character set in a standard way. "$" is meant to mean "dollars" and not some special notation to help the system designers through a rough spot.
- Definitions are maintained for every language token in the system. These definitions can be recalled at will by the user. The entire dictionary, with definitions, is also available.

Efficient systems architecture

The "MUSE" system has incorporated many efficiencies of structure:

- Modular system construction is used for across-computer-implementation and ease of upgrading capabilities.
- Assembly language coding permits economies of size and speed.
- The "MUSE" dialogs are, in fact, applications programs for the sake of reproducibility of output.
- As a subsystem, "MUSE" borrows features of the time-sharing system which condenses its size.
- Advanced system programming techniques are used throughout.

CONCLUSION

The general objective of "MUSE" has been to enhance the time-shared computer environment by providing a natural language for machine-user communication. It is designed to provide the manager with a medium of interaction with a large common data base, loaded and maintained by the "MUSE" system.

"MUSE" is capable of performing as a simulation model builder, statistical tool, data screening and sorting aid, and report organizer for non-routine, creative