AFIPS CONFERENCE PROCEEDINGS

1980 NATIONAL COMPUTER CONFERENCE

May 19-22, 1980

Anaheim, California

DON MEDLEY
Editor and Program Chairman

ELLEN MARIE RANDALL
Editorial/Production Specialist

HERBERT SAFFORD
Conference Chairman

AFIPS PRESS
1815 NORTH LYNN STREET
ARLINGTON, VIRGINIA 22209

Programming with data frames for everyday data items*

by DAVID W. EMBLEY
University of Nebraska
Lincoln, Nebraska

INTRODUCTION

Processing everyday data items such as dollar amounts, time, dates, and account numbers constitutes a significant portion of real-world computer applications. Programmers involved with everyday data items confront the drudgery of writing routines to recognize, validate, transform, store, retrieve, manipulate, and display these items and also the challenge to develop user-friendly data-entry systems and insure data integrity. They usually meet these challenges using various and sundry ad hoc techniques.

Sometimes, much of the burden is transferred to data-entry personnel who are asked to adhere to rigid input formats and to insure accuracy by tedious double checking. As explained by Gilb and Weinberg in their book *Humanized Input,* many poor system designs have been saved by the accurate touch of the keypunch operators.[1]

In an attempt to systematically address the problems encountered when processing everyday data items, the concept of a *data frame* is proposed. A data frame provides a means to encapsulate the concept of a data item with all of its essential properties including alternative natural language written forms, computer representation, applicable contextual information, permissible operations, and relationships with other data items.

DATA FRAMES

The name "data frame" has been coined because of the concept's similarity to data abstractions[2,3] and Minsky frames.[4] A data frame can be thought of as an extension to data abstractions or as a Minsky frame cast in the form of an abstract data type.

Minsky's theory of frames is a theory of rich symbolic structure where a frame represents a particular situation. Included in the frame is information about how to use the knowledge, what can be expected, and what to do if expectations are not confirmed. Data frames represent data items instead of situations, but the information included and its purpose are quite similar.

An attempt to precisely define the syntactic structure that includes all information that might be needed for all possible applications is premature, but it seems reasonable to extend the structure of data abstractions to represent the additional information required. The aim is to appropriately model the behavior of a particular data item.

Figure 1 shows some of the essential features for a data frame for dollar amounts. There are several acceptable written forms for U. S. currency, for example, \$25.63, \$2,-638,457.00, \$.63, 63¢, \$47, 47\$ or just plain 25.63 when the context is understood. In general, the dollar amount input routine should accept any of these forms and perform the necessary translation to an internal computer representation. A return code supplied by the input routine gives the completion status and provides information for error messages. The corresponding output routine produces a formatted string ready to be displayed. A more detailed description of input/output conversions is given elsewhere.[5]

The context keywords in Figure 1 are extracted from the forms in common use in the Computer Science Department at the University of Nebraska. In the context of one of these keywords, if a data item is expected, it is quite certain that the type of the data item is a dollar amount. Also extracted from the forms are the operations addition, subtraction, and multiplication by an integer, along with context keywords that indicate an operator's applicability.

A library of commonly used data frames would be a valuable asset to application programmers who could then extract, possibly modify, and use them along with data frames created by themselves to fit their needs. A general dollar amount data frame taken from a library, for instance, could be adopted for use in an application involving UNL Computer Science departmental forms with very little if any alteration. The general library version might have a somewhat longer or different list of context keywords and an additional operator or two, but would be essentially the same.

Data frames can also be grouped together to model items more complex than a data element. Indeed, a data frame group is indistinguishable from a data frame for an elementary item except that selector operations would be available to provide access to an item within the group. A date, for instance, may be constructed as an elementary data frame or as a data frame group with selector functions for day, month, and year.

* This material is based upon work supported by the National Science Foundation under Grant No. MCS-7904126.

```
data frame dollar_amount;

internal representation real;

input (s: string) returns (dollar_amount, return_code);
     (* validate string s and translate it into the internal representation *)
     ...
     end;

output (a: dollar_amount, f: format specification) returns (string, return_code);
     (* translate a dollar amount variable a into a string according to a
       given format specification f *)
     ...
     end;

context keywords:  amount, amt, budget, cost, dollar, expense, extension, funds,
     honorarium, income, price, rate, sal, salary, sales, total, $;

infix function (a1: dollar_amount) + (a2: dollar_amount)
     returns (dollar_amount);
     context keywords: add, total, +;
     return (a1+a2);
     end;

infix function (a1: dollar_amount) - (a2: dollar_amount)
     returns (dollar_amount);
     context keywords: difference, subtract, -;
     return (a1-a2);
     end;

infix function (a: dollar_amount) * (n: integer) returns (dollar_amount);
     context keywords: amount, extension;
     return (a*n);
     end;

end.
```

Figure 1—Illustration of some of the essential features for a dollar amount data frame

In the next three sections, examples are presented to show how data frame features might be utilized.

Computer-assisted instruction example

One of the problems with CAI is the lack of flexibility in accepting student responses. The main reason for this rigidity is that input validation and interpretation is so difficult. For questions that require common items of data as answers, data frames can help make the development of user-friendly input requirements easier.

Figure 2 shows several student responses to the question, "When was Abraham Lincoln born?", and a reasonable system reply to each response based on information from a data frame for dates. A date input routine would reject "Kentucky" and "32 Feb, 1890" as invalid dates. Both "2/12/1809" and "12/2/1809" would be acceptable because the ambiguity could be resolved by comparing with the expected answer. By making use of the available date operations, dates that are too early, late, or only partially correct could also be identified.

```
When was Abraham Lincoln born?

        > Kentucky

          No, the answer should be a valid date.
          Try again.

        > 32 Feb 1809

          No, the answer should be a valid date.
          Try again.

        > 2/12/1809

          Ok, your answer is correct.  (February 12, 1809)

        > 12/2/1809

          Ok, your answer is correct.  (February 12, 1809)

        > November 17, 1805

          No, he was born after that.
          Try again.

        > Jan. 4, 1815

          No, he was born before that.
          Try again.

        > 12-2-1808

          No, the year is wrong.
          Try again.

        > 12 February 1809

          Ok, your answer is correct.
```

Figure 2—Student responses (preceded by ">") and CAI-system replies to a question requiring a date for an answer

Query language example

At the University of Nebraska, several students and faculty in the Department of Computer Science together with personnel from the Computing Network are in the process of developing SIMPLE, a programming environment designed for instructional use for beginning students and students with minimal system requirements.[6] As part of the project, a database has been established containing information about student users such as account number, name, account balance, processor usage, and available command set. A query language was hastily provided to enable instructors to access and update the information, but a more user-friendly interface based on data-frame ideas is being designed.

The data-frames query language has no syntactic structure, neither is it based on natural language processing concepts. Instead, self-identifying data in its natural language form, context keywords, and available operators provide the information necessary for inferring the desired action.

Figure 3 shows a sample dialogue between a class instructor and the system. In the first request, the "$25" can be recognized as a dollar amount; then, with the context keyword " + " and the lack of any account designation, the system proposes that all accounts receive an additional 25 dollars. In the second request, the English phrase should not be construed to carry meaning; the presence of the keyword "ADD" along with a dollar amount and account numbers is all that is necessary. In the third request, a context dependency arises since the instructor has just singled out two specific accounts. In a second attempt, the discrepancy is resolved. For this request, of course, a data frame on SIMPLE commands would have to be available to recognize GET, PUT, and APPEND as commands and to recognize ADD as a context keyword associated with set union. (SIMPLE allows an instructor to tailor its use by specifying which commands are available to each user.) The only reasonable response to a name when no context information applies is to display information about the individual, and since the database is not being altered, no confirmation is necessary. (In SIMPLE all command keywords are uniquely identified by their first letter.) In the fifth request, SINCE is not recognized, so more than one reasonable alternative exists. If an impasse is reached where the system sees no other reasonable alternatives, the query language provides a structured request format that can be used as a last resort.

Forms-based programming example

In a forms-based approach to programming, an application programmer describes data processing operations by means of structures modeled on conventional administrative forms. For a particular function, the programmer designs the form, specifies the data type for each item on the form, and defines relationships among the items. A system of related forms can be defined so that information is manipulated and routed among forms, a database, and external devices.[7]

```
>+$25
ADD $25 TO ALL ACCOUNTS? (Y/N)
>Y
DONE

>ADD AN ADDITIONAL $10 TO ACCOUNTS CSC0321 AND CSC0607.
ADD $25 TO CSC0321 CSC0607? (Y/N)
>Y
DONE

>ADD GET, PUT, AND APPEND
INCLUDE GET PUT APPEND IN COMMAND SET FOR CSC0321 CSC0607? (Y/N)
>N
INCLUDE GET PUT APPEND IN COMMAND SET FOR ALL ACCOUNTS? (Y/N)
>Y
DONE

>MARK MEYER
ACCOUNT #  NAME        BALANCE   NR. RUNS   COMMANDS
CSC0097    MARK MEYER  $43.17    12         ABCDFGHILNOPQRT

>NUMBER OF PASCAL RUNS FOR ALL USERS SINCE 12 APR
ON APRIL 12, 1979? (Y/N)
>N
FROM APRIL 12, 1979 TO PRESENT? (Y/N)
>Y
493
```

Figure 3—SIMPLE query language (lines preceded by ">" are input by the user)

Suppose that the invoice-voucher form shown in Figure 4 is part of a system of forms that models a business operation. As an application programmer designs the form, blank spaces for expected entries are specified to be of a certain type and are associated with key phrases that indicate what entries are expected. The blank spaces for vendor name, street, city, state, zip code, and voucher total $ all expect a single entry. Item no., description, quantity, unit price, and amount are members of a group of items each containing an unknown but equal number of entries.

With a library of appropriate data frames at its disposal, it is conceivable that a forms-based programming system could select the appropriate frame for each item on the form and define relationships among the items without any intervention from an application programmer. For the dollar amount entries, the associated key phrases all contain context keywords that appear in the data frame in Figure 1. It is not hard to imagine that the other needed data frames likewise contain the essential context keywords to enable the entry type to be determined.

In addition to context keywords, data frames used for forms-based programming also require knowledge about the nature and layout of forms. The addition operator in the dollar amount data frame, for instance, must know that a column of dollar amounts can be added together to produce a total. For the invoice-voucher, this information coupled with the expectation of a total in the blank space immediately below the column of dollar amounts is enough for the system to propose the relationship $\Sigma \text{amount}_i = \text{voucher total } \$$.

Rows also imply relationships. Record information is often placed on a row; and therefore, if the item no., description, and unit price match field descriptors for records in a database file, values for these entries could be either double checked against a database or partially acquired from the file given only the item no.

The relationship $\text{quantity}_i * \text{unit price}_i = \text{amount}_i$ is also de-

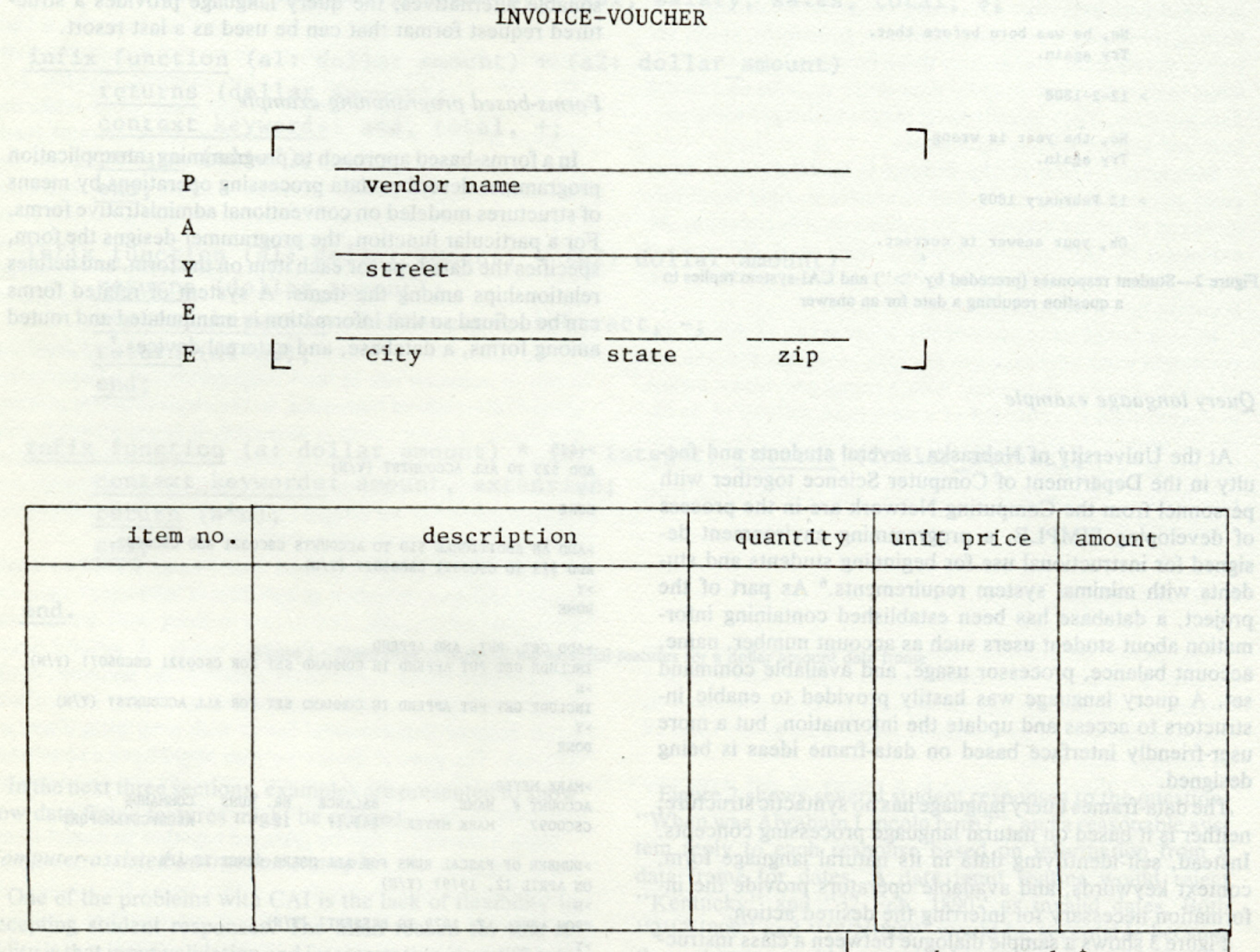

Figure 4—Invoice-voucher form

ducible from the facts at hand. Where relationships cannot be deduced or where ambiguities or misunderstandings arise, an application programmer can supply the information or resolve the issue and always has the final say.

CONCLUDING REMARK

Although much remains to be done, even a step toward the encapsulation of the essential properties of everyday data items in data frames would benefit programmers in many applications. Extending data abstractions with input/output routines alone could simplify many programming tasks particularly for business and interactive computing applications where handling input and output constitutes a significant portion of the programming effort.

REFERENCES

1. Gilb, T. and Weinberg, G. M., *Humanized Input,* Winthrop Publishers, 1977.
2. Guttag, J. V., Horowitz, E., and Musser, D. R., "The Design of Data Type Specifications," in *Current Trends in Programming Methodology,* R. T. Yeh (ed.), Prentice-Hall, 1978, pp. 60-79.
3. Liskov, B. H. and Zilles, S. N., "Programming with Abstract DataTypes," *Proceedings of ACM Symposium on Very High Level Languages, SIGPLAN Notices,* Vol. 9, No. 4, April 1974, pp. 50-59.
4. Minsky, M., "A Framework for Representing Knowledge," in *The Psychology of Computer Vision,* P. H. Winston (ed.), McGraw-Hill, 1975, pp. 211-277.
5. Embley, D. W., *Data Abstractions for Everyday Data Items,* Department of Computer Science, University of Nebraska-Lincoln, August 1979.
6. Embley, D. W. and Nagy, G., *SIMPLE Specifications,* Department of Computer Science, University of Nebraska-Lincoln, June, 1979.
7. Embley, D. W., "Forms-Based Automatic Program Generation," *ACM 78 Proceedings,* December 1978, pp. 972-979.